When building or running an application, the environment variables PATH and LD_LIBRARY_PATH may be used to find the commands and/or runtime shared libraries needed for the operation. If multiple entries in these variables contain commands or libraries with the same filename as the one needed, this can potentially lead to conflicts.

The system searches the PATH or LD_LIBRARY_PATH entries from beginning to end, and always uses the first matching entry it finds —regardless of whether it is the correct one. Using the wrong command or shared library could cause your application to fail or provide incorrect results. Therefore, it is very important to pay attention to which entries are included in PATH and LD_LIBRARY_PATH, and where they are placed.

PATH and LD_LIBRARY_PATH Entries

Some default directories (such as /usr/local/bin and /usr/bin) are automatically included in PATH for every NAS user account. Many users modify PATH in their system startup files (such as .cshrc, .login, .profile, or .bashrc). For LD_LIBRARY_PATH, a system default is not defined; users typically define LD_LIBRARY_PATH in their own scripts.

Software modules add entries to PATH and LD_LIBRARY_PATH when they are loaded. The placement of an entry depends on two factors:

- Whether a modulefile uses prepend-path or append-path.

  For example, the tecplot/2017r2 modulefile prepends the /nasa/tecplot/2017r2/360ex_2017r2/bin:/nasa/tecplot/2017r2/chorus_2017r2/bin entry to the beginning of the existing PATH, while the netcdf/4.4.1.1_serial modulefile appends the /nasa/netcdf/4.4.1.1_serial/bin entry to the end of the PATH.

  To find out whether prepend-path or append-path is used by a given modulefile, run:

  ```
  module show modulefile_name
  ```

- The order in which the modulefiles are loaded.

  Entries are added to PATH and LD_LIBRARY_PATH in the order the modulefiles are loaded. For example, both the hdf5/1.8.18_serial and pkgsrc/2016Q4 modulefiles use prepend-path to add their corresponding bin directories to PATH. If hdf5/1.8.18_serial is loaded before pkgsrc/2016Q4, the entries from pkgsrc will be listed first, as follows:

  ```
  /nasa/pkgsrc/sles12/2016Q4/bin:/nasa/pkgsrc/sles12/2016Q4/sbin:/nasa/hdf5/1.8.18_serial/bin
  ```

Known Conflicts

It is a common problem that users load a long list of modules for their entire workflow in their system startup scripts, but are not aware that some of these modules contain commands and/or shared libraries that are in conflict with one another. A few known conflicts are described below.

NAS pkgsrc Module

The NAS-built pkgsrc module contains nearly 1,000 packages. Its bin and lib directories contain many commands and libraries that conflict with the bin and lib directories in the system default locations or other software modules. For example:

- The cmake command is included in both the /usr/bin and /nasa/pkgsrc/sles12/2016Q4/bin directories.
- The libhdf5.so library is included in the /nasa/pkgsrc/sles12/2016Q4/lib directory, the /nasa/hdf5/1.8.18_serial/lib directory, and the /nasa/hdf5/1.8.18_mpt/lib directory.

Other Software Packages

Some commercial or third-party software packages are self-contained and might include duplicate commands or libraries that are available in other modulefiles. For example:

- The Tecplot package comes with its own Intel libraries, such as libimf.so, libifcore.so, and libifport.so. If your LD_LIBRARY_PATH includes both Tecplot's library directories and the Intel compiler's library directories, your application could fail in unpredictable ways—or worse, it could silently return incorrect results.
- Many commercial packages (such as matlab, ansys, flow3d, ansa, nastran, star-ccm, mathematica, and so on) provide the libiomp5.so library in their distributions. Loading one of these software modules together with an Intel compiler module might cause problems for OpenMP applications.

  For example, running the OVERFLOW MPI+OpenMP hybrid code results in the following error when a matlab module is loaded after an Intel compiler module:

  ```
  overflowmpi: relocation error: overflowmpi: symbol kmp_aligned_malloc, version VERSION not defined in file libiomp5.so with link time reference
  ```

  This occurs because the matlab modulefile prepends its library path to LD_LIBRARY_PATH, so that entry is listed first. The libiomp5.so library needed by OVERFLOW is therefore found in the matlab distribution's entry (wrong copy) instead of in the Intel distribution's entry (correct copy).

Best Practices

To avoid software conflicts, follow these best practices:

- Load modules in a working session and/or in PBS scripts, instead of in your shell startup scripts.
- Do not load the pkgsrc module unless it is needed.
- Load only the modules that are needed by the application you are currently using.
- When in doubt, check which command or library you are using.

  To find the directory containing the command you are using, run:

  which *command_name*

  To find the paths to shared libraries satisfied by the command you are using, run:

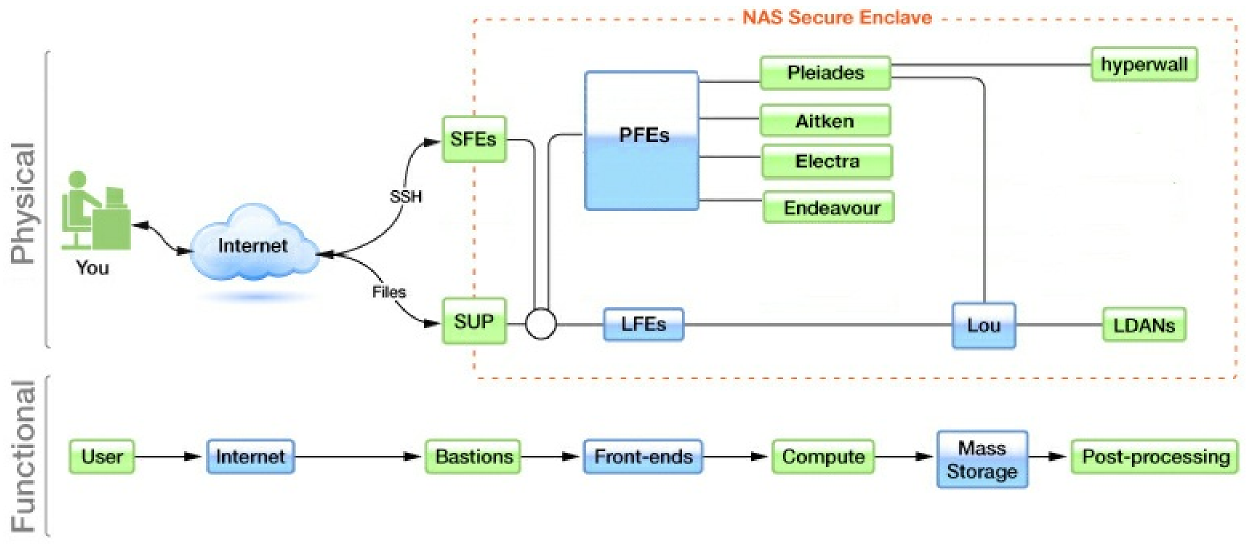
  ldd *directory/command_name*

Our HPC environment is operated by staff in the NASA Advanced Supercomputing (NAS) Division at Ames Research Center located at Moffett Field, CA. The supercomputers and support staff are funded by NASA's High-End Computing Capability (HECC) Project.

The topics in this section summarize the system components, such as the compute nodes, secure network connections, front-end systems, and data storage facilities; and the user environment, including information about the Linux operating system and the various filesystem directories that are available for your use.

Systems Overview

The systems available for your use are all protected within a secure environment. The figure below shows both the physical connections among all the components and their functional relationships through a typical user workflow. The environment is described in more detail in the following sections.



The Secure Enclave

NAS supercomputing components are protected within a secure enclave that can be accessed only by authenticated users through the following secure bastions:

- Secure front ends (SFEs)
- Secure Unattended Proxy (SUP)

Components protected within the secure enclave include:

- Supercomputers: Pleiades, Aitken, Electra, Endeavour
- Front-end nodes: Pleiades, Aitken, Electra, Endeavour (PFEs), Lou (LFEs)
- Lou mass storage system
- Lou data analysis nodes (LDANs)
- hyperwall visualization system

The following sections give an overview of each component of the secure enclave.

Bastions

Secure Front Ends (SFEs)

The secure front ends (SFEs) provide inbound connection from your local system to the secure enclave. The first time you access the HECC systems within the enclave, you will authenticate through an SFE. Subsequently, you can use any of the bastions to access systems in the enclave.

For an overview of the initial authentication process, see Logging in for the First Time. For more information about the SFEs, see the article Role of the Secure Front Ends.

Secure Unattended Proxy (SUP)

The Secure Unattended Proxy (SUP) allows you to pre-authenticate to the secure enclave for one-week periods, during which you can perform unattended (batch) file transfers. After you complete the setup process, SUP is the most efficient and convenient method for transferring files from your remote system. For more information, see Using the Secure Unattended Proxy (SUP).

Front Ends

The HECC supercomputers Pleiades, Aitken, Electra, and Endeavour share the Pleiades front-end systems (PFEs). You can use the PFEs to edit files, compile your code, run short debugging and testing sessions, and submit batch jobs to the Pleiades, Aitken, or Electra compute nodes or to Endeavour. See the following articles for more information:

- Pleiades Front-End Usage Guidelines
- Pleiades Front-End Load Balancer

Compute Nodes

There are currently four supercomputers available for users: Pleiades, Aitken, Electra, and Endeavour.

Pleiades

NASA's flagship supercomputer, and one of the most powerful production systems in the world, Pleiades is an HPE/SGI ICE cluster containing multiple generations of Intel processors. See the following articles for more information:

- Pleiades Resource Page
- Pleiades configuration and usage guidelines

Aitken

Aitken, NASA's newest supercomputer, is housed in the Modular Supercomputing Facility, an environmentally-friendly module located a short distance from the main NAS building. Aitken uses the Pleiades front ends (PFEs), filesystems, PBS server, and job queues. See the following articles for more information:

- Aitken Resource Page
- Aitken Configuration Details
- Preparing to Run on Aitken Cascade Lake Nodes
- Preparing to Run on Aitken Rome Nodes

Electra

Electra is NASA's first prototype modular supercomputing system, housed near the main NAS building. Electra uses the Pleiades front ends (PFEs), filesystems, PBS server, and job queues. See the following articles for more information:

- Electra Resource Page
- Electra Configuration Details
- Preparing to Run on Electra Skylake Nodes
- Preparing to Run on Electra Broadwell Nodes

Endeavour

Endeavour is an HPE Superdome Flex system that provides resources for user applications needing access to large cache-coherent, global shared-memory capabilities in a single system image (SSI). Endeavour uses the Pleiades front ends (PFEs) and filesystems, and shares some of the Pleiades InfiniBand fabric. However, Endeavour uses its own designated Portable Batch System (PBS) server and job queues. See the following articles for more information:

- Endeavour Resource Page
- Endeavour Configuration Details
- Preparing to Run on Endeavour

Mass Storage System

The NAS facility provides long-term storage space on a single mass storage system, known as Lou. This HPE/SGI system has 7.6 petabytes (PB) of disk space and is capable of storing up to 1040 PB (1 exabyte) on tape. See the following articles for more information:

- Your Mass Storage Directory
- Lou Mass Storage System

Post-Processing Systems

Systems provided for post-processing include the Lou data analysis nodes (LDANs), designated as ldan[*11-14*], and the hyperwall visualization system. For a summary on using these systems for post-processing work, see Post-Processing Your Data. For detailed information, see also the following articles:

- Pleiades Front-End Usage Guidelines
- Lou Data Analysis Nodes
- Visualization System: hyperwall

Networks

The NAS high-speed network (NASLAN) includes a 10 gigabit-per-second (Gb/s) local area network and 10 Gb/s peering with other high-speed networks such as the NASA Integrated Communications Services (NICS), Internet2, and the Consortium for Educational Networks in California (CENIC). For an overview, see Networking Resources.

To access the HECC resources inside the secure enclave, you will use the SSH protocol to connect from your desktop system to a bastion (usually the SFEs) through a wide area network and the NASLAN.

User Environment

All HECC systems run the Linux operating system. If you are new to Linux, you can find a lot of helpful information at the user-supported community website Linux.org, including a Beginners Learning Course that provides instruction on the basic directory structure of Linux, how to get around in the directories, how to access Linux documentation (man pages), useful commands, and much more. You can also find support at the Linux forum.

When your NAS account is created, your default Linux shell is set to be the C shell (csh); this is assumed to be the case throughout this guide. If you want to use a different shell as your default, such as bash, call the NAS Control Room staff at (800) 331-8737 or (650) 604-4444 or send an email message to support@nas.nasa.gov to request the change. After the change is made, the new default shell of your choice applies to all of your jobs.

Once you complete the initial setup for your NAS account, you will have access to the Pleiades front-end (PFE) systems, the home filesystems, the Lou mass storage filesystems, and the scratch (/nobackup) filesystems. Your NAS account is authorized to run jobs on all HECC compute systems.

NAS supercomputers use the Portable Batch System (PBS) from Altair Engineering, Inc., for job submission, monitoring, and management. For more information about PBS, see Submitting and Running Jobs.

Filesystems

Pleiades, Aitken, Electra, and Endeavour share the same home and scratch (/nobackup) filesystems. When you log into a PFE, you will have access to the following directories:

- A home directory on the Pleiades home filesystem, which you can use to store a small number of files such as source code, small input files, and so on
- A /nobackup directory on a Lustre filesystem, which you can use to temporarily store larger files for reading and writing large amounts of data while running jobs

For long-term data storage, you also have access to a home directory on the Lou mass storage systems. The /nobackup filesystems are mounted on Lou, so you can easily copy files there directly from your /nobackup directory.

The Pleiades and Lou home filesystems are backed up each night. These backups are stored for approximately one year. The scratch (/nobackup) filesystems are *not* backed up.

Quota limits are enforced on all filesystems. Two kinds of quotas are supported:

- Limits on the total disk space occupied by your files
- Limits on the number of files you can store, irrespective of size; for quota purposes, directories count as files

See Quota Policy on Disk Space and Files for more information.

Your Home Directory

Your home directory is located on the Pleiades home filesystem, which is accessible from Pleiades, Aitken, Electra, and Endeavour. Use your home directory to store a limited number of smaller files such as source code and input files. For temporary, short-term storage of larger files, use your /nobackup directory. For long-term data storage, use the Lou mass storage systems. See Pleiades Home Filesystem for more information.

Your Scratch (/nobackup) Directory

Use your /nobackup directory to temporarily store large files that read and write large amounts of data when you run jobs. Your /nobackup directory resides on one of several Lustre filesystems, designated /nobackupp*X*. To find out which Lustre filesystem your /nobackup directory is located on, run:

```
pfe21% ls -ld /nobackup/your_nas_username
```

The /nobackup filesystems are also mounted on the Lou mass storage system, so you can access data in your /nobackup directory from Lou without going through Pleiades, Aitken, Electra, or Endeavour.

WARNING: The /nobackup filesystems mean just that: they are not backed up. While this is stating the obvious, some users have lost important data by storing it on these systems over long periods of time. It is your responsibility to copy essential data to either your home directory, to archival storage on the Lou systems, or to your remote system.

See Pleiades Lustre Filesystems for more information.

Your Mass Storage Directory

For safe, long-term data storage, transfer your files to your Lou home directory. The Lou mass storage filesystem allows you to retrieve your stored files quickly and securely whenever you need them.

You can log into the Lou system just as you would any other HECC system and save data to mass storage by copying your files to your Lou home directory: Lou:/u/*your_nas_username*.

There is no specified data-size quota for your Lou home directory, but you can store up to 250,000 files.

For more information, see The Lou Mass Storage System.

The Data Migration Facility

Data stored on Lou is migrated to tape, as needed, to make space on the disks for more data. Migrated files are retrieved to active disk when you attempt to read or write to them. These migration and retrieval processes are managed by HPE/SGI's Data Migration Facility (DMF), which also enables you to manually list, put, find, and get files that are on tape.

When your data is migrated to tape, two copies are written to two separate tape media in automated tape libraries located in two different buildings. See Data Migration Facility Commands for more information.

The Lou Filesystem

Lou is composed of the Lou front ends (LFEs), designated lfe[*5-8*]. The /nobackup filesystems are mounted on the LFEs, so you can easily copy files there directly from your /nobackup directory.

Although you cannot perform post-processing tasks on the LFEs, the Lou data analysis nodes provide PBS resources to perform post-processing tasks on your Lou mass storage data. For more information, see the following articles:

- The Lou Mass Storage System
- Lou Data Analysis Nodes

Software Modules and Packages

HECC provides many software programs such as compilers, pre- and post-processing programs, analysis tools, and math and scientific libraries. The software is managed through the use of packages and modules that you can load into your home directories.

For more information, see the following articles:

- Software on NAS Systems
- Customizing Your UNIX Environment

Taking the time to customize settings in your NAS environment will help make your workflow more convenient and efficient. This article covers some common user customizations that can be included in your startup configuration files, which contain information that is read by the Linux shell every time you log into the computer and/or open a new terminal window.

Common Shells and Their Startup Files

The current default shell is bash (unless you chose a different one in your NAS account request form). If you want csh to be your default, send a request to support@nas.nasa.gov.

bash
: The startup file .profile is sourced when you log in. The file .bashrc is sourced when you start an interactive non-login shell.

csh
: The startup files .cshrc followed by .login are sourced when you log in. The file .cshrc is sourced every time you start a new shell.

Upon your first login to a Pleiades front end (PFE), you can find the .profile, .login, and .cshrc files with default contents provided by NAS. You can modify them, or create new ones from scratch.

For the customization described below, you can choose which files to modify based on what changes you want to take effect.

With csh, it is common to include the following line in the .cshrc file to check whether a prompt exists:

```
if (! $?prompt) exit #exit if not interactive
```

Important: If you want your customization to take effect for both interactive sessions (such as a login shell, new shell, or an interactive PBS shell) and non-interactive sessions (such as a PBS batch job), be sure to add them *above this line*.

Set Permission for New Files and Directories with umask

```
umask 077
```

Sets the permission of all new files and/or directories to be read/write/execute by owner only.

```
umask 037
```

Sets the permission to be read/write/execute by owner and readable by group members.

Load Software Modules

To use modules, be sure to include the following line in your startup configuration file:

- For bash: source /usr/local/lib/global.profile
- For csh: source /usr/local/lib/global.cshrc

No default software is loaded on NAS systems. If you want to automatically load some software when you log in, such as compilers and MPI library, use the module load command in your startup configuration file. For example:

```
module load comp-intel mpi-hpe
```

See Software on NAS Systems for more information.

Augment $PATH

If you normally use certain directory paths to find commands, scripts, or executables, you can add them to your path in your startup configuration file. You can add other directories to the path, such as the /bin directory under your $HOME, and the /u/scicon/tools/bin directory, where many useful, recommended tools are installed:

For bash: PATH=$PATH:$HOME/bin:/u/scicon/tools/bin

For csh: set path = ($path ~/bin /u/scicon/tools/bin)

Set Aliases

Aliases allow you to replace one string with another string. For example, if you add the following line in your startup configuration file, the command ls -lrt is used whenever you type ls:

For bash: alias ll='ls -lrt'

For csh: alias ll 'ls -lrt'

Set Environment Variables

Set environment variables that you need regularly for the current shell and sub-shells. For example:

For bash: export CFDROOT=/u/*your_nas_username*/bin

For csh: setenv CFDROOT /u/*your_nas_username*/bin

Filesystems

A common way to share files and/or directories with group members or others is to use the chmod command to change the permissions. However, chmod has limitations, so you may sometimes choose to use Access Control Lists (ACLs).

When you issue the command chmod g+rx *filename*, for example, all the members in your group (g) gain read (r) and search/execute (x) access to that file, as shown below:

```
% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% chmod g+rx filename

% ls -l filename
-rw-r-x--- 1 zsmith s0101 9 Jun 10 12:11 filename
```

However, chmod does not allow you to select which members of your group or which specific individuals outside of your group can access your files/directories. ACLs provide a mechanism for greater control of file sharing. There are two ACL commands:

setfacl

Set file access control lists.

```
SYNOPSIS
    setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...

    setfacl --restore=file
```

A detailed usage explanation of setfacl and its options can be found via **man setfacl**. Among the options listed:

1. The -m or -M option lets you "modify" the ACL, where -m expects an ACL on the command line and -M expects an ACL from a file or from standard input
2. The -x or -X option removes the ACL entries
3. The -R or --recursive option applies operations to all files and directories recursively
4. The --test option allows you to test the effect of changing the ACL without actually changing it
5. The -b option removes all extended ACL entries except the base entries of the owner, group, and others

getfacl

Get file access control lists.

```
SYNOPSIS
    getfacl [-dRLPvh] file ...

    getfacl [-dRLPvh] -
```

A detailed usage explanation of getfacl and its options can be found via **man getfacl**.

Note: Before you grant another user or group access to certain files or directories, make sure that access to the parent directory (where the files or directories reside) is also allowed.

Example 1

To allow another user (*jbrown*) to have read/execute (rx) permission on a file (*filename*) and to view the ACL before and after an ACL change:

```
% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
group::---
other::---

% setfacl -m u:jbrown:rx filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
```

```
user:jbrown:r-x
group::---
mask::r-x
other::---

% ls -l filename
-rw-r-x---+ 1 zsmith s0101 9 Jun 10 12:11 filename
```

Example 2

To remove all extended ACLs in Example 1 except the base entries of the owner, group, and others:

```
% setfacl -b filename

% ls -l filename
-rw------- 1 zsmith s0101 9 Jun 10 12:11 filename

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
group::---
other::---
```

Example 3

Continuing from Example 1, to test the granting of read/execute (rx) access to another group (group id 24176) without actually doing it:

```
% setfacl --test -m g:24176:rx filename
filename: u::rw-,u:jbrown:r-x,g::---,g:g24176:r-x,m::r-x,o::---,*

% getfacl filename
# file: filename
# owner: zsmith
# group: s0101
user::rw-
user:jbrown:r-x
group::---
mask::r-x
other::---
```

Example 4

To allow another user (*jbrown*) recursive access to a directory (dir.abc which contains a file *filename*):

```
% ls -ld dir.abc
drwx------ 2 zsmith s0101 17 Jun 10 13:19 dir.abc

% ls -l dir.abc
total 0
-rw------- 1 zsmith s0101 0 Jun 10 13:19 filename

% setfacl -R -m u:jbrown:rx dir.abc

% getfacl dir.abc
# file: dir.abc
# owner: zsmith
# group: s0101
user::rwx
user:jbrown:r-x
group::---
mask::r-x
other::---

% getfacl dir.abc/filename
# file: dir.abc/filename
# owner: zsmith
# group: s0101
user::rw-
user:jbrown:r-x
group::---
mask::r-x
other::---

% ls -ld dir.abc
drwxr-x---+ 2 zsmith s0101 17 Jun 10 13:19 dir.abc
```

```
% ls -l dir.abc
total 0
-rw-r-x---+ 1 zsmith s0101 0 Jun 10 13:19 filename
```

Example 5

Continuing from Example 4, to recursively remove all permissions user *jbrown* for a directory:

```
% setfacl -R -x u:jbrown dir.abc

% getfacl dir.abc
# file: dir.abc
# owner: zsmith
# group: s0101
user::rwx
group::---
mask::---
other::---

% getfacl dir.abc/filename
# file: dir.abc/filename
# owner: zsmith
# group: s0101
user::rw-
group::---
mask::---
other::---
```

For more information on ACLs, read **man acl**.

Using 'Giveto' to Copy Files and/or Directories to Another User

NAS's in-house developed giveto script is built on the use of Access Control Lists (ACLs). It allows one user (the giver) to copy files and/or directories to a /nobackup directory of another user (the recipient).

giveto is installed under /usr/local/bin on Pleiades and Lou.

In the example below, user zsmith gives a copy of his dir.abc directory on Pleiades to user jbrown. The steps describe the giveto command used by each of them, and the results.

1. User *jbrown* uses the command giveto -i *zsmith* to automatically (a) create an INCOMING directory (if it does not already exist) under her /nobackup/*jbrown* and (b) grant user *zsmith* read/write/execute permission on this directory.

```
pfe21:/u/jbrown% giveto -i zsmith
nobackup[1] =  /nobackup/jbrown

pfe21:/u/jbrown% ls -ld /nobackup/jbrown/INCOMING
drwxrwx---+ 2 jbrown s0202 4096 Jun 14 12:18 /nobackup/jbrown/INCOMING
```

2. User *zsmith* uses the command giveto *jbrown* dir.abc to automatically (a) create a subdirectory called *zsmith*_0 under *jbrown*'s INCOMING directory, (b) copy dir.abc to /nobackup/*jbrown*/INCOMING/*zsmith*_0, (c) grant user jbrown read/write/execute permission on /nobackup/*jbrown*/INCOMING/*zsmith*_0, and (d) send an email to user *jbrown* regarding the copy.

```
pfe21:/home1/zsmith> ls -ld dir.abc
drwx------ 2 zsmith s0101 17 Jun 14 12:21 dir.abc/

pfe21:/home1/zsmith> giveto jbrown dir.abc
setfacl -m u:jbrown:rwx zsmith_0
setfacl -m u:jbrown:rwx zsmith_0/giveto.log
setfacl -m u:jbrown:rwx zsmith_0/dir.abc
setfacl -m u:jbrown:rwx zsmith_0/dir.abc/foo2
path =  /nobackup/jbrown/INCOMING/zsmith_0
total 12
drwxrwx---+ 3 zsmith  s0101 4096 Jun 14 12:29 .
drwxrwx---+ 3 jbrown  s0202 4096 Jun 14 12:29 ..
drwxrwx---+ 2 zsmith  s0101 4096 Jun 14 12:21 dir.abc
-rw-rwx---+ 1 zsmith  s0101   44 Jun 14 12:29 giveto.log
```

   Note: If the directory *zsmith*_0 already exists prior to this step, *zsmith*_1 would be used instead.

3. User *jbrown* receives an email from user *zsmith* with a subject line "giveto files". They see that the directory dir.abc has been copied successfully. Even though the directory /nobackup/*jbrown*/INCOMING/*zsmith*_0 is still owned by user *zsmith*, user *jbrown* now has permission to read/write/execute files and directories under /nobackup/*jbrown*/INCOMING/*zsmith*_0.

```
pfe21:/u/jbrown% ls -lrt /nobackup/jbrown/INCOMING
total 4
drwxrwx---+ 3 zsmith s0101 4096 Jun 14 12:29 zsmith_0

pfe21:/u/jbrown%ls -lrt /nobackup/jbrown/INCOMING/zsmith_0
total 4
drwxrwx---+ 2 zsmith s0101 4096 Jun 14 12:21 dir.abc

pfe21:/u/jbrown%ls -lrt /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
total 4
-rw-rwx---+ 1 zsmith s0101 8 Jun 14 12:21 foo2

pfe21:/u/jbrown%getfacl /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
# file: /nobackup/jbrown/INCOMING/zsmith_0/dir.abc
# owner: zsmith
# group: s0101
user::rwx
user:jbrown:rwx
group::---
mask::rwx
other::---
```

Read **man giveto** for more information.

The giveto script was created by NAS staff member Arthur Lazanoff.

Quota Policy on Disk Space and Files

Filesystems on Pleiades and Lou have the following types of quotas:

- Limits on the total disk space occupied by your files
- Limits on the number of files (represented by inodes) you can store, regardless of size. For quota purposes, directories count as files.

Hard and Soft Quota Limits

NAS quotas have hard limits and soft limits. Hard limits should never be exceeded. Soft limits can be exceeded temporarily, for a grace period of 14 days. If your data remains over the soft limit for more than 14 days, the soft limit is enforced as a hard limit. On some filesystems, this means that write attempts will fail. On other filesystems, it will mean that your PBS jobs will no longer start. To reduce your data to below the quota limits, you can delete unneeded files or copy important files elsewhere, such as the Lou mass storage system, and then remove them locally.

Filesystem quotas are shown in the following table:

	Pleiades	**Lou**
$HOME	NFS	XFS
Space: soft	8 GB	none
Space: hard	10 GB	none
Inode: soft	none	250,000
Inode: hard	none	300,000
/nobackup	Lustre /nobackupp*X*	N/A
Space: soft	1 TB	N/A
Space: hard	2 TB	N/A
Inode: soft	500,000	N/A
Inode: hard	600,000	N/A

To learn how to check your disk space, inode usage, and quotas, see the following articles:

- Pleiades Home Filesystem
- Pleiades Lustre Filesystems
- The Lou Mass Storage System

Email Warnings and Consequences

It is expected that your data will exceed your soft limits as needed. When this occurs, you will begin to receive daily emails to inform you of your current disk space and how much of your grace period remains. However, if your data is still over the soft limit or if you reach the hard limit, restrictions are put in place.

The following table shows the quota enforcement policy for each type of filesystem.

Filesystem	**Grace Period**	**Enforcement**
/nobackupp*X*	2 weeks	Batch jobs won't run
/nobackupnfs2	2 weeks	Writes are disabled
Pleiades home directory	2 weeks	Writes are disabled
Lou home directory	2 weeks	Writes are disabled

More details are available in the following sections.

Lustre Filesystems (/nobackupp*X*)

If your grace period has expired, then your batch queue access is restricted and no new jobs can be run until your data on the Lustre filesystem is reduced to an amount below the soft limit. Any jobs that are running will continue to run. If you reach the hard limit, your batch access is immediately restricted, but any running jobs will continue.

Pleiades Home Directory and /nobackupnfs2

If your grace period has expired, then any writes to that filesystem will fail, and any jobs that attempt to write to the filesystem will fail. You will be unable to write to the filesystem until your data is reduced to an amount below the soft limit. Similarly, if you reach the hard limit, any writes will immediately fail and you will be unable to write to the filesystem until your data is reduced to an amount below the hard limit.

Lou Home Directory

If your grace period has expired, then any writes that attempt to create new files will fail. You will be unable to create any new files until the number of files is reduced to below the soft limit. If you reach the hard limit, any writes that attempt to create new files will fail until the number of files is reduced to below the hard limit.

The maximum size of a tar file moved to Lou should not exceed 2 TB. If you need to archive larger files, please contact the NAS Control Room at support@nas.nasa.gov for assistance.

Note: The Shift tool, which is a convenient way to transfer files to Lou, can created a set of smaller independent tar files from a single large directory. For more information, see Shift File Transfer Overview.

Changing Your Quotas

If an account needs larger quota limits, send an email justification tosupport@nas.nasa.gov. Your request will be reviewed by management for approval.

Pleiades Home Filesystem

The home filesystems on Pleiades are HPE/SGI NEXIS 9000 filesystems that are NFS-mounted on all of the Pleiades front-end (PFE) nodes and compute nodes. Each NAS user is provided a home directory on the Pleiades home filesystems, which are the home filesystems for Pleiades, Aitken, Electra, and Endeavour. After you are granted an account, your home directory is set up automatically during your first login.

Your home directory has a quota of 8-10 GB of storage. For temporary storage of larger files, use the Lustre /nobackup filesystems. For long-term storage, use the Lou mass storage system.

Quota Limits and Policy

Disk space quota limits are enforced on the Pleiades home filesystems. By default, the soft limit is 8 GB and the hard limit is 10 GB. There are no inode limits on the home filesystem.

To check your quota and usage on your home filesystem, run the `quota -v` command as follows:

```
%quota -v
Disk quotas for user username (uid xxxx):
    Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
saturn-ib1-0:/mnt/home2
               7380152  8000000 10000000          190950       0       0
```

The NAS quota policy states that if you exceed the soft quota (the number listed under `quota` in the above sample output), an email will be sent to inform you of your current usage and how much of the grace period remains. It is expected that you will occasionally exceed your soft limit; however, after 14 days, any attempts to write to the filesystem will result in error. Any attempt to exceed the hard limit (the number listed under `limit` in the above sample output) will result in error.

If you believe that you have a long-term need for higher quota limits on the Pleiades home filesystem, send an email justification to support@nas.nasa.gov. Your request will be reviewed by the HECC Deputy Project Manager for approval.

Note: For temporary storage of larger files, or a large number of files, use your Lustre /nobackupp*X* directory. For normal long-term file storage, transfer your files to the Lou mass storage systems.

See also: Quota Policy on Disk Space and Files.

TIP: If you receive the following error message when logging in, you won't be able to run X applications. This error is usually due to exceeding your home filesystem quota. To avoid the error, decrease your disk usage.

```
/usr/X11R6/bin/xauth: error in locking authority file /u/username/.Xauthority
```

Backup Schedule

Files on the home filesystem are backed up daily.

Pleiades Lustre Filesystems

Pleiades, Aitken, Electra, and Endeavour share several filesystems intended to provide working space for compute jobs. These filesystems, called "nobackup" (/nobackupp*X*), provide over 90 petabytes (PB) of disk space and serve many thousands of cores. The /nobackup filesystems are Lustre-based filesystems managed under Lustre software version 2.*x*.

Using the Lustre /nobackup Filesystems

As the name suggests, these "nobackup" filesystems are for temporary use, and are not backed up. Lustre can handle many large files, but you cannot use the Lustre filesystems for long-term storage. If you want to save your files, move them to Lou.

Each Lustre filesystem is shared among many users. To learn how to achieve good I/O performance for your applications and avoid impeding the I/O operations of other users, read the related articles listed at the bottom of the page.

Lustre filesystem configurations are summarized at the end of this article.

WARNING: Any files that are removed from Lustre /nobackup filesystems *cannot* be restored. You should store essential data on the Lou mass storage system (lfe[*5-8*]) or on other, more permanent storage.

Which Lustre Filesystem is Assigned to Me?

Once you are granted a NAS account, one of the Lustre filesystems will be assigned to you. To find out which one, run thels command as follows:

```
pfe21% ls -l /nobackup/your_username
```

In the output, look for the filesystem name (nobackupp*X*). For example, the following output shows that the user's assigned filesystem is /nobackupp17:

```
lrwxrwxrwx 1 root root 19 Sep 23  2021 /nobackup/your_username -> /nobackupp17/your_username
```

Default Quota and Policy on /nobackup Filesystems

Disk space and inode quotas are enforced on the /nobackup filesystems. The default soft and hard quota limits for inodes are 500,000 and 600,000, respectively. Quotas for the disk space are 1 terabyte and 2 terabytes, respectively. To check your disk space, inode usage, and quota on your /nobackup filesystem, run the lfs command as follows:

```
% lfs quota -h -u username /nobackup/username
Disk quotas for user username (uid nnnn):
      Filesystem     used      quota   limit   grace   files    quota   limit   grace
/nobackup/username       4k      1.024T  2.048T       -       1    500000  600000       -
```

It is expected that your data will occasionally exceed the soft limit. However, after a 14-day grace period, your access to the batch queues will be restricted and you will not be able to run new jobs until your data is reduced below the soft limit. If you reach the hard limit, your batch access will immediately be restricted, but any running jobs will continue.

If you anticipate a long-term need for higher quota limits, please send a justification via email tosupport@nas.nasa.gov. Your request will be reviewed by the HECC Deputy Project Manager for approval.

For more information, see Quota Policy on Disk Space and Files.

Note: When a Lustre filesystem is full, the jobs writing to it will hang. A Lustre error with code -28 in the system log file indicates that the filesystem is full. NAS support staff will typically send emails to those using the most space, with a request to clean up their files.

Lustre Filesystem Configurations

The way your Lustre filesystem is configured determines how your files are striped. There are two types of configurations: Progressive File Layout (PFL), and the standard Lustre configuration. Most of the /nobackup filesystems have PFL configurations.

Progressive File Layout Configurations

Lustre's Progressive File Layout (PFL) feature enables filesystems to dynamically change the stripe count for files based on their size. This means that the files are dynamically striped, therefore, you should ***not*** set custom stripe size or stripe counts.

The Lustre PFL filesystems are /nobackupp[*10-29*]; among them, /nobackupp[*17-19, 27-29*] are equipped with a small number of solid state drives (SSDs) in addition to the hard disk drives (HDDs), while /nobackupp[*10-13,15-16*] only have HDDs.

The HDD configurations of /nobackupp[*10-29*] are listed in the table below with the abbreviation nbp*X*. P = petabytes; T = terabytes; M = megabytes.

Lustre PFL HDD Configurations

Filesystem	nbp10	nbp11	nbp12,13,15	nbp17,18	nbp19	nbp27,28	nbp29

# of OSTs	46	69	23	32	16	32	16
size/OST	70.7 T	70.7 T	70.7 T	565 T	565 T	565 T	565 T
Total Space	3.2 P	4.8 P	1.6 P	18.6 P	9.3 P	18.6 P	9.3 P

For /nobackupp[*10-13,15-16*], which have only HDDs, a common default PFL stripe setting is used as follows:

Lustre PFL Stripe Counts per File Size

File Size	0 - 10 MB	10 MB - 17 GB	17 - 68 GB	> 68 GB
Stripe Count	1	4	8	16

For /nobackupp[*17-19, 27-29*], different PFL configurations are set among the SSD and HDD pools. To learn more, see Progressive File Layout with SSD and HDD Pools.

Standard Configurations

The standard Lustre filesystems are /nobackupp1 and 2; in the table below, these are abbreviated as nbp*X*. P = petabytes; T = terabytes; M = megabytes.

Standard Lustre Configurations

Filesystem	nbp1	nbp2
# of OSTs	144	342
size/OST	43.6/57.2 T	43.6/71.2 T
Total Space	7.1 P	18 P
Default Stripe Size	1 M	1 M
Default Stripe Count	4	4

A Lustre filesystem is a high-performance shared filesystem for Linux clusters that is managed with Lustre software. It is highly scalable and can support many thousands of client nodes, petabytes of storage, and hundreds of gigabytes per second of I/O throughput. The NAS Lustre filesystems are named "/nobackupp*X*."

Each Lustre filesystem is actually a set of many small filesystems, which are referred to as object storage targets (OSTs). The Lustre software presents the OSTs as a single unified filesystem.

For more information about OSTs and other Lustre filesystem components, see Main Lustre Components.

Useful Lustre Commands

The examples in this section use /nobackupp*17* as an example of a specific Lustre filesystem.

Listing Disk Usage and Quotas

To display disk usage and limits on your /nobackup directory:

```
pfe21% lfs quota -h -u username /nobackupp17
```

or

```
pfe21% lfs quota -h -u username /nobackup/username
```

To display usage on each OST, add the -v option:

```
pfe21% lfs quota -h -v -u username /nobackup/username
```

Listing Space Usage

To list space usage per OST and MDT, in human-readable format, for all Lustre filesystems or for a specific one:

```
pfe21% lfs df -h
pfe21% lfs df -h /nobackupp17
```

Listing Inode Usage

To list inode usage for all filesystems or for a specific one:

```
pfe21% lfs df -i
pfe21% lfs df -i /nobackupp17
```

Listing OSTs

To list all the OSTs for the filesystem:

```
pfe21% lfs osts /nobackupp17
```

Viewing Striping Information

To view the striping information for a specific file or directory:

```
pfe21% lfs getstripe filename
pfe21% lfs getstripe -d directory_name
```

Note: Omitting the -d flag will display striping for all files in the directory.

File Striping

Files on the Lustre filesystems are striped automatically. This means they are transparently divided into chunks that are written or read simultaneously across a set of OSTs within the filesystem. The chunks are distributed among the OSTs using a method that ensures load balancing. Files larger than 100 gigabytes (GB) *must* be striped in order to avoid taking up too much space on any single OST, which might adversely affect the filesystem.

Benefits include:

- Striping allows one or more clients to read/write different parts of the same file at the same time, providing higher I/O bandwidth to the file because the bandwidth is aggregated over the multiple OSTs.
- Striping allows file sizes larger than the size of a single OST.

However, striping small files increases the number of objects in each file, resulting in increased overhead due to network operations—sometimes creating server contention. This is not likely to cause problems for files larger than 4 megabytes (MB), but in many cases, striping can cause noticeably slower read/write performance for files smaller than 4 MB.

Progressive File Layout

With the deployment of the new Lustre progressive file layout (PFL) feature, different stripe settings for different segments of a file can be configured in such a way that a small stripe count is used for the beginning segment and larger stripe counts are used for latter segments as the file grows. The newer Pleiades Lustre filesystems, /nobackupp[*10-29*], have all been configured with NAS-selected PFL settings.

Important: In most cases, you should rely on these defaults and not manually set file striping yourself. The information in the following sections are useful when using the older, non-PFL filesystems, /nobackupp[*1-2*].

To learn more about filesystems configured with PFL, see Progressive File Layout with SSD and HDD Pools.

Selecting a Stripe Count

The default stripe count on /nobackupp[*1-2*] filesystems is currently 4. The following examples describe circumstances where it is beneficial to change the stripe count to a different number:

- Your program reads a single large input file, where many nodes read or write different parts of the file at the same time. You should stripe this file adequately to prevent all the nodes from reading from the same OST at the same time. This will avoid creating a bottleneck in which your processes try to read from a single set of disks.
- You have other large files. You should restripe large files adequately to keep capacity balanced across the OSTs and maintain consistent performance for all users.
- Your program waits while a large output file is written. You should stripe the large file so that it will write faster, in order to reduce the amount of time the processors are idle.
- Your program periodically writes several small files from each processor. It is not usually necessary to have a stripe count greater than 1 for small files, and they will be randomly distributed across the OSTs to maintain good performance and capacity balance in aggregate.

Setting Stripe Parameters

There are default stripe configurations for each Lustre filesystem. To check the existing stripe settings on your directory, use the lfs getstripe command as follows:

```
pfe21% lfs getstripe -d /nobackup/username
```

In most cases, the only tuning you need to do is to change the number of OSTs that your files are written to. To change the number of OSTs, use the lfs setstripe command as follows:

```
pfe21% lfs setstripe -c stripe_count dir|filename
```

Note: The stripe settings of an existing file cannot be changed. If you want to change the settings of a file, create a new file with the desired settings and copy the existing file to the newly created file.

Examples of Striping

Newly created files and directories inherit the stripe settings of their parent directories. You can take advantage of this feature by organizing your large and small files into separate directories, then setting a stripe count on the large-file directory so that all new files created in the directory will be automatically striped. For example, to create a directory called "restart" with a stripe count of 8, run:

```
pfe21% mkdir restart
pfe21% lfs setstripe -c 8 restart
```

You can "pre-create" a file as a zero-length striped file by running lfs setstripe as part of your job script or as part of the I/O routine in your program. Then, you can write to that file later. For example, to pre-create the file "big_dir.tar" with a stripe count of 20, and then add data from the large directory "big_dir," run:

```
pfe21% lfs setstripe -c 20 big_dir.tar
pfe21% tar cf big_dir.tar big_dir
```

Advanced Parameters

The following stripe parameters are rarely needed, but could potentially be useful, depending on your application I/O:

Stripe size: -S
: Sets the size of the chunks in bytes. Use with k, m, or g to specify units of KB, MB, or GB, respectively (for example, -S 2m). The specified size must be a multiple of 65,536 bytes (64 KB). The default size is 1 MB for all Pleiades Lustre filesystems; specify 0 to use the default.

Stripe offset: -o
: Sets the index of the OST where the first stripe is to be placed. The default is -1, which results in random selection. Using a non-default value is *not* recommended.

See the **lfs man page** for more options and information.

Main Lustre Components

Lustre filesystem components include:

Metadata Server (MDS)
: Service nodes that manage all metadata operations, such as assigning and tracking the names and storage locations of directories and files on the OSTs. There is 1 MDS per filesystem.

Object Storage Target (OST)
: Storage devices where users' file data is stored. The size of each OST varies from approximately 7 terabytes (TB) to approximately 22 TB, depending on the Lustre filesystem. The capacity of a Lustre filesystem is the sum of the sizes of all OSTs. There are multiple OSTs per filesystem.

Metadata Target (MDT)
: A storage device where the metadata (name, ownership, permissions and file type) are stored. There is 1 MDT per filesystem.

Object Storage Server (OSS)
: Service nodes that run the Lustre software stack, provide the actual I/O service and network request handling for the OSTs, and coordinate file locking with the MDS. Each OSS can serve multiple OSTs. The aggregate bandwidth of a Lustre filesystem can approach the sum of bandwidths provided by the OSSes. There can be 1 or multiple OSSes per filesystem.

Lustre Clients
: Compute nodes that mount the Lustre filesystem, and access/use data in the filesystem. There are commonly thousands of Lustre clients per filesystem.

Hard disk drives (HDDs) and solid state drives (SSDs) are the two main storage options commonly deployed in data centers. With HDD, data are stored magnetically in spinning disks, and data retrieval is slow; therefore, HDD is practical only for data that does not need to be accessed frequently. SSDs are a newer technology where data is stored in integrated circuits, which dramatically reduces access time. Because SSDs are more expensive than HDDs, a hybrid solution provides a good compromise between price and performance.

Six of the NAS Lustre filesystems, /nobackupp[*17-19, 27-29*], are configured with Progressive File Layout (PFL) with HDDs and SSDs configured as different Lustre pools. This article provides information about these filesystems using /nobackupp17 as an example.

SSD and HDD Pools

To view a list of a filesystem's object storage targets (OSTs) showing whether they are in the HDD pool or the SSD pool, use the lfs pool_list command. The OSTs are shown in hexadecimal format:

```
pfe% lfs pool_list nbp17.hdd-pool
Pool: nbp17.hdd-pool
nbp17-OST0000_UUID
nbp17-OST0001_UUID
..
nbp17-OST001e_UUID
nbp17-OST001f_UUID

pfe% lfs pool_list nbp17.ssd-pool
Pool: nbp17.ssd-pool
nbp17-OST0064_UUID
nbp17-OST0065_UUID
...
nbp17-OST0082_UUID
nbp17-OST0083_UUID
```

The available SSD space in each filesystem is much smaller than the HDD space. In the example below, the lfs df command shows that the size of each /nobackupp17 HDD OST is 565.0 terabytes (TB), while each SSD OST is only 30.3 TB. The output also shows both the hexadecimal (far left) and decimal (far right) labels of each OST.

```
pfe% lfs df -h /nobackupp17 | grep OST
nbp17-OST0000_UUID      565.0T    322.7T     213.8T  61% /nobackupp17[OST:0]
nbp17-OST0001_UUID      565.0T    323.5T     213.0T  61% /nobackupp17[OST:1]
...
nbp17-OST001e_UUID      565.0T    322.9T     213.7T  61% /nobackupp17[OST:30]
nbp17-OST001f_UUID      565.0T    325.1T     211.4T  61% /nobackupp17[OST:31]
nbp17-OST0064_UUID       30.3T     23.5T       6.4T  79% /nobackupp17[OST:100]
nbp17-OST0065_UUID       30.3T     23.5T       6.4T  79% /nobackupp17[OST:101]
...
nbp17-OST0082_UUID       30.3T     23.5T       6.5T  79% /nobackupp17[OST:130]
nbp17-OST0083_UUID       30.3T     23.5T       6.5T  79% /nobackupp17[OST:131]
```

Configuration of the /nobackupp[*17-19, 27-29*] Filesystems

The following table summarizes the configurations of /nobackupp[*17-19, 27-29*].

Filesystem	**nbp17**	**nbp18**	**nbp19**	**nbp27**	**nbp28**	**nbp29**
# of OSTs in HDD	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)	32 (OST: 0-31)	32 (OST: 0-31)	16 (OST: 0-15)
Size/OST in HDD	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T	565.0 T
Total Space in HDD	18.6 P	18.6 P	9.3 P	18.6 P	18.6 P	9.3 P
# of OSTs in SSD	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)	32 (OST: 100-131)	16 (OST: 100-115)	16 (OST: 100-115)
Size/OST in SSD	30.3 T	60.5 T	29.6 T	30.3 T	60.5 T	29.6 T
Total Space in SSD	~ 1 P	~ 1 P	~ 0.5 P	~ 1 P	~ 1 P	~ 0.5 P

Progressive File Layout (PFL)

The /nobackupp[*17-19, 27-29*] filesystems take advantage of the Lustre PFL feature, which increases the stripe count of each file in a step-wise manner as the file grows in size. Specifically, PFL is built using composite layouts described by a series of components. Each component covers a non-overlapping portion of the file and has its own stripe setting in size, count, index and pool.

Default PFL Configurations

For each filesystem, a NAS-created default layout provides reasonable performance for a variety of file I/O patterns—and relieves users of the need to determine and set striping.

To find the default layout of each filesystem, use the lfs getstripe command as follows:

```
pfe% lfs getstripe -d /nobackupp17
lcm_layout_gen:    0
...
 lcme_extent.e_start: 0
 lcme_extent.e_end:   268435456
  stripe_count:  1    stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 268435456
 lcme_extent.e_end:   5368709120
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  ssd-pool

 ...
 lcme_extent.e_start: 5368709120
 lcme_extent.e_end:   EOF
  stripe_count:  16   stripe_size: 16777216   pattern:   raid0   stripe_offset: -1  pool:  hdd-pool
```

The output shows that /nobackupp17 is configured with three components:

- The first component setting applies to the file segment from the beginning up to 268435456 B (256 MB), with a stripe count of 1 and stripe size of 16 MB; this component uses ssd-pool.
- The second component setting applies to the file segment from 256 MB to 5 GB with a stripe count of 16 and stripe size of 16 MB; this component uses ssd-pool.
- The last component setting applies to the file segment from 5 GB to the end of the file with a stripe count of 16 and stripe size of 16 MB; this component uses hdd-pool.

With this composite layout, I/O to and from the first 5 GB of files will benefit from the higher performance of SSDs.

Note: The default component layouts may be adjusted by NAS system administrators as needed.

Customized PFL Configurations

If you have experience using the PFL feature, and you want to have more control of your application's I/O, you can choose your own composite layouts using the lfs setstripe command with various options, such as --component-end (or -E), --stripe-size (or -S), --stripe-count (or -c), --overstripe-count (or -C), --pool (or -p). For example:

```
lfs setstripe \
   -E 256M -C 1  -S 1M -p ssd-pool \
   -E 4G   -C 4  -S 1M -p ssd-pool \
   -E EOF  -C 16 -S 1M -p hdd-pool \
directory_name_or_filename
```

Note: If not explicitly set, files inherit layouts from their parent directory.

WARNING: SSD space is limited. Therefore, do not set striping for very large files to use only the SSD pool.

Mirroring and Migration of Data from SSD to HDD

As the limited SSD space fills up, data that is stored in SSDs is migrated over to HDDs to free up SSD space for I/O by new jobs. This automatic migration is accomplished through a mirroring mechanism.

After a file is closed, SSD data is queued to be mirrored to the HDD. The data will be in dual state—stored on both the SSD and the HDD—until SSD usage reaches a certain point; then, the SSD copy will be removed to free up space. Typically, the oldest files are removed first when SSD usage reaches 80%, and other files are removed more aggressively when usage reaches 90%. The mirroring and migration settings may be adjusted by NAS system administrators as needed.

Note: Mirrored data that exists in both SSDs and HDDs are counted twice against your quota.

To find out if a file is in dual state or has been migrated completely to HDDs, use the lfs getstripe filename command, as shown in the next two examples.

Output Showing Data in Dual State

If the lfs getstripe filename output shows entries with different values for lcme_mirror_id, the data is in dual state, as illustrated in this example:

```
pfe% lfs getstripe file1
file1
  lcm_layout_gen:    5
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1     <-------
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 128
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 128, l_fid: [0x800000409:0x413e0a:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1   <--------
    lcme_flags:          init,prefer
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   5368709120
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 103
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 103, l_fid: [0x340000409:0x4305bb:0x0] }
      - 1: { l_ost_idx: 131, l_fid: [0x500000409:0x449f9f:0x0] }
      .....
      - 14: { l_ost_idx: 120, l_fid: [0x9c000040a:0x415384:0x0] }
      - 15: { l_ost_idx: 127, l_fid: [0x7c0000409:0x4100a6:0x0] }

    lcme_id:             65539
    lcme_mirror_id:      1  <---------
    lcme_flags:          init
    lcme_extent.e_start: 5368709120
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 27
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 27, l_fid: [0xfc0000400:0xd887b2:0x0] }
      - 1: { l_ost_idx: 26, l_fid: [0xb00000404:0xd8ed7c:0x0] }
      ....
      - 14: { l_ost_idx: 23, l_fid: [0xe40000400:0xd97afb:0x0] }
      - 15: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9a4:0x0] }

    lcme_id:             131073
    lcme_mirror_id:      2  <--------
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 9
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 9, l_fid: [0xe00000400:0x36d9bd:0x0] }
      - 1: { l_ost_idx: 19, l_fid: [0x120000040d:0xdaa1a0:0x0] }
      ....
      - 14: { l_ost_idx: 28, l_fid: [0xf40000400:0xd7c343:0x0] }
      - 15: { l_ost_idx: 11, l_fid: [0xc00000402:0x36d1eb:0x0] }
```

Output Showing Data Completely Migrated to HDDs

This example shows an older file that has been completely migrated to HDDs:

```
pfe% lfs getstripe file2
file2
  lcm_layout_gen:    2
  lcm_mirror_count:  1
  lcm_entry_count:   1
    lcme_id:             131073
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0        <-----
    lcme_extent.e_end:   EOF      <-----
      lmm_stripe_count:  16
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 24
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 24, l_fid: [0xd8000040e:0x475b02:0x0] }
      - 1: { l_ost_idx: 3, l_fid: [0x130000040e:0x4733cd:0x0] }
      ....
      - 14: { l_ost_idx: 20, l_fid: [0xc8000040e:0x482bcc:0x0] }
      - 15: { l_ost_idx: 18, l_fid: [0xe8000040e:0x47cc62:0x0] }
```

Note: When your file is completely migrated to HDD (removed from SDD), the (ctime) of the file will change to the creation time of the mirrored copy on HDD, as described below.

Change Time (ctime) Example

When a file is mirrored, a second copy of the data is created. The file will retain both sets of data until you break the mirror, or when the filesystem automatically flushes the SSD pool—at which time, the (ctime) will be updated to reflect the time the mirror was created.

In the following example, the stat command is used to show the time metadata of a test file that was created at 15:48:45:

```
% stat testfile
  File: 'testfile'
  Size: 59535       Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
```

At time 15:50:27, the file is mirrored but the ctime remains unchanged:

```
% stat testfile
  File: 'testfile'
  Size: 59535       Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:48:45.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:48:45.000000000 -0800
 Birth: -

pfe25 /nobackupp18/username % lfs getstripe testfile
testfile
  lcm_layout_gen:    1
  lcm_mirror_count:  2
  lcm_entry_count:   4
    lcme_id:             65537
    lcme_mirror_id:      1
    lcme_flags:          init,prefer
    lcme_extent.e_start: 0
    lcme_extent.e_end:   268435456
      lmm_stripe_count:  1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 109
      lmm_pool:          ssd-pool
      lmm_objects:
      - 0: { l_ost_idx: 109, l_fid: [0x90000040a:0x53990:0x0] }

    lcme_id:             65538
    lcme_mirror_id:      1
    lcme_flags:          prefer
    lcme_extent.e_start: 268435456
    lcme_extent.e_end:   5368709120
      lmm_stripe_count:  -1
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          ssd-pool

    lcme_id:             65539
    lcme_mirror_id:      1
    lcme_flags:          0
    lcme_extent.e_start: 5368709120
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  16
      lmm_stripe_size:   16777216
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: -1
      lmm_pool:          hdd-pool

    lcme_id:             131073
    lcme_mirror_id:      2
    lcme_flags:          init
    lcme_extent.e_start: 0
    lcme_extent.e_end:   EOF
      lmm_stripe_count:  1
      lmm_stripe_size:   1048576
      lmm_pattern:       raid0
      lmm_layout_gen:    0
      lmm_stripe_offset: 0
      lmm_pool:          hdd-pool
      lmm_objects:
      - 0: { l_ost_idx: 0, l_fid: [0x2c0000409:0x30c3e:0x0] }
```

Later, the file is flushed from the SSD. At that time, the file's `ctime` is updated to reflect the creation time of the copy on HDD:

```
% stat testfile
  File: 'testfile'
  Size: 59535       Blocks: 120        IO Block: 4194304 regular file
Device: 521e79ech/1377729004d Inode: 342273918398214817  Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2022-02-14 15:50:27.000000000 -0800
Modify: 2022-02-14 15:48:45.000000000 -0800
Change: 2022-02-14 15:50:27.000000000 -0800
 Birth: -
```

References

- Progressive File Layout
- File Level Redundancy
- Layout Enhancement High Level Design

At NAS, Lustre (/nobackup) filesystems are shared among many users and many application processes, which can cause contention for various Lustre resources. This article explains how Lustre I/O works, and provides best practices for improving application performance.

How Does Lustre I/O Work?

When a client (a compute node from your job) needs to create or access a file, the client queries the metadata server (MDS) and the metadata target (MDT) for the layout and location of the file's stripes. Once the file is opened and the client obtains the striping information, the MDS is no longer involved in the file I/O process. The client interacts directly with the object storage servers (OSSes) and object storage targets (OSTs) to perform I/O operations such as locking, disk allocation, storage, and retrieval.

If multiple clients try to read and write the same part of a file at the same time, the Lustre distributed lock manager enforces coherency so that all clients see consistent results.

Jobs being run on Pleiades contend for shared resources in NAS's Lustre filesystem. Each server that is part of a Lustre filesystem can only handle a limited number of I/O requests (read, write, stat, open, close, etc.) per second. An excessive number of such requests, from one or more users and one or more jobs, can lead to contention for storage resources.. Contention slows the performance of your applications and weakens the overall health of the Lustre filesystem. To reduce contention and improve performance, please apply the examples below to your compute jobs while working in our high-end computing environment.

Best Practices

Avoid Using ls -l

The ls -l command displays information such as ownership, permission, and size of all files and directories. The information on ownership and permission metadata is stored on the MDTs. However, the file size metadata is only available from the OSTs. So, the ls -l command issues RPCs to the MDS/MDT and OSSes/OSTs for every file/directory to be listed. RPC requests to the OSSes/OSTs are very costly and can take a long time to complete if there are many files and directories.

- Use ls by itself if you just want to see if a file exists
- Use ls -l *filename* if you want the long listing of a specific file

Avoid Having a Large Number of Files in a Single Directory

Opening a file keeps a lock on the parent directory. When many files in the same directory are to be opened, it creates contention. A better practice is to split a large number of files (in the thousands or more) into multiple subdirectories to minimize contention.

Avoid Accessing Small Files on Lustre Filesystems

Accessing small files on the Lustre filesystem is not efficient. When possible, keep them on an NFS-mounted filesystem (such as your home filesystem on Pleiades /u/*username*) or copy them from Lustre to /tmp on each node at the beginning of the job, and then access them from /tmp.

Keep Copies of Your Source Code on the Pleiades Home Filesystem and/or Lou

Be aware that files under /nobackup are *not* backed up. Make sure that you save copies of your source codes, makefiles, and any other important files on your Pleiades home filesystem. If your Pleiades home directory quota isn't large enough to keep all of these files, you can request a larger quota and/or create tarred copies of these files on Lou.

Avoid Accessing Executables on Lustre Filesystems

There have been a few incidents on Pleiades where users' jobs encountered problems while accessing their executables on the /nobackup filesystem. The main issue is that the Lustre clients can become unmounted temporarily when there is a very high load on the Lustre filesystem. This can cause a bus error when a job tries to bring the next set of instructions from the inaccessible executable into memory.

Executables run slower when run from the Lustre filesystem. It is best to run executables from your home filesystem on Pleiades. On rare occasions, running executables from the Lustre filesystem can cause executables to be corrupted. Avoid copying new executables over existing ones of the same name within the Lustre filesystem. The copy causes a window of time (about 20 minutes) where the executable will not function. Instead, the executable should be accessed from your home filesystem during runtime.

Limit the Number of Processes Performing Parallel I/O

Given that the numbers of OSSes and OSTs on Pleiades are about a hundred or fewer, there will be contention if a large number of processes of an application are involved in parallel I/O. Instead of allowing all processes to do the I/O, choose just a few processes to do the work. For writes, these few processes should collect the data from other processes before the writes. For reads, these few processes should read the data and then broadcast the data to others.

Understand the Effect of Stripe Counts/Sizes for MPI Collective Writes

For programs that call MPI collective write functions, such as MPI_File_write_all, MPI_File_write_at_all, and MPI_File_write_ordered, it is important to understand the effect of stripe counts on performance.

Background

MPI I/O supports the concept of collective buffering. For some filesystems, when multiple MPI processes are writing to the same file in a coordinated manner, it is much more efficient for the different processes to send their writes to a subset of processes in order to do a smaller number of bigger writes. By default, with collective buffering, the write size is set to be the same as the stripe size of the file.

With Lustre filesystems, there are two main factors in the SGI MPT algorithm that chooses the number of MPI processes to do the writes: the stripe count and number of nodes. When the number of nodes is greater than the stripe count, the number of collective buffering processes is the same as the stripe count. Otherwise, the number of collective buffering processes is the largest integer less than the number of nodes that evenly divides the stripe count. In addition, MPT chooses the first rank from the first *n* nodes to come up with *n* collective buffering processes.

Note: Intel MPI behaves similarly to SGI MPT on Lustre filesystems.

Enabling Collective Buffering Automatically

You can let each MPI implementation enable collective buffering for you, without any code changes.

SGI MPT automatically enables collective buffering for the collective write calls using the algorithm described above. This method requires no changes in the user code or in the mpiexec command line. For example, if the stripe count is 1, only rank 0 does the collective writes, which can result in poor performance. Therefore, experimenting with different stripe counts on the whole directory and/or individual files is strongly recommended.

Intel MPI also does collective buffering, similar to SGI MPT, when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
       -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
       -np xx a.out
```

Enabling Collective Buffering via Code Changes

In this method, you provide "hints" in the source code to inform MPI what to do with specific files. For example:

```
  call MPI_Info_create(info, status)
  call MPI_Info_set(info, "romio_cb_write", "enable", STATUS)
  call MPI_Info_set(info, "striping_unit", "1048576", STATUS)
  call MPI_Info_set(info, "striping_factor", "16", STATUS)
  ...
  call MPI_File_open(MPI_COMM_WORLD, file_name, MPI_MODE_WRONLY, info, unit, status)
```

Note: The hints are only advisory and may not be honored. For example, SGI MPT 2.12r26 honors these hints, but MPT 2.14r19 does not. Intel MPI 5.0x honors these hints when the I_MPI_EXTRA_FILESYSTEM and I_MPI_EXTRA_FILESYSTEM_LIST variables are set appropriately, as follows:

```
mpiexec.hydra  -env I_MPI_EXTRA_FILESYSTEM on \
       -env  I_MPI_EXTRA_FILESYSTEM_LIST lustre \
       -np xx a.out
```

Stripe Align I/O Requests to Minimize Contention

Stripe aligning means that the processes access files at offsets that correspond to stripe boundaries. This helps to minimize the number of OSTs a process must communicate for each I/O request. It also helps to decrease the probability that multiple processes accessing the same file communicate with the same OST at the same time.

One way to stripe-align a file is to make the stripe size the same as the amount of data in the write operations of the program.

Avoid Repetitive "stat" Operations

Some users have implemented logic in their scripts to test for the existence of certain files. Such tests generate "stat" requests to the Lustre server. When the testing becomes excessive, it creates a significant load on the filesystem. A workaround is to slow down the testing process by adding sleep in the logic. For example, the following user script tests the existence of the files WAIT and STOP to decide what to do next.

```
touch WAIT
 rm STOP
 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
 end
```

When neither the WAIT nor STOP file exists, the loop ends up testing for their existence as quickly as possible (on the order of 5,000 times per second). Adding sleep inside the loop slows down the testing.

```
touch WAIT
 rm STOP

 while ( 0 <= 1  )
  if(-e WAIT) then
    mpiexec ...
    rm WAIT
  endif
  if(-e STOP) then
    exit
  endif
  sleep 15
 end
```

Avoid Having Multiple Processes Open the Same File(s) at the Same Time

On Lustre filesystems, if multiple processes try to open the same file(s), some processes will not able to find the file(s) and your job will fail.

The source code can be modified to call the sleep function between I/O operations. This will reduce the occurrence of multiple, simultaneous access attempts to the same file from different processes.

```
100  open(unit,file='filename',IOSTAT=ierr)
      if (ierr.ne.0) then
       ...
      call sleep(1)
      go to 100
      endif
```

When opening a read-only file in Fortran, use ACTION='read' instead of the default ACTION='readwrite'. The former will reduce contention by not locking the file.

```
open(unit,file='filename',ACTION='READ',IOSTAT=ierr)
```

Avoid Repetitive Open/Close Operations

Opening files and closing files incur overhead and repetitive open/close should be avoided.

If you intend to open the files for read only, make sure to use ACTION='READ' in the open statement. If possible, read the files once each and save the results, instead of reading the files repeatedly.

If you intend to write to a file many times during a run, open the file once at the beginning of the run. When all writes are done, close the file at the end of the run.

See Lustre Basics for more information.

Use the Soft Link to Refer to Your Lustre Directory

Your /nobackup directory is created on a specific Lustre filesystem, such as /nobackupp17 or /nobackupp18, but you can use a soft link to refer to the directory no matter which filesystem it is on:

/nobackup/*your_username*

By using the soft link, you can easily access your directory without needing to know the name of the underlying filesystem. Also, you will not need to change your scripts or re-create any symbolic links if a system administrator needs to migrate your data from one Lustre filesystem to another.

Preserve Corrupted Files for Investigation

When you notice a corrupted file in your /nobackup directory, it is important to preserve the file to allow NAS staff to investigate the cause of corruption. To prevent the file from being accidentally overwritten or deleted by your scripts, we recommend that you rename the corrupted file using:

```
pfe% mv filename filename.corrupted
```

Note: Do not use cp to create a new copy of the corrupted file.

Report the problem to NAS staff by sending an email to support@nas.nasa.gov. Include how, when, where the corrupted file was generated, and anything else that may help with the investigation.

Best Practices for Non-PFL Filesystems

Important: The tips in this section apply only to /nobackupp[*1-2*].

Use a Stripe Count of 1 for Directories with Many Small Files

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

If you must keep small files on Lustre, be aware that stat operations are more efficient if each small file resides in one OST. Create a directory to keep small files in, and set the stripe count to 1 so that only one OST will be needed for each file. This is useful when you extract source and header files (which are usually very small files) from a tarfile. Use the Lustre utility lfs to create a specific striping pattern, or find the striping pattern of existing files.

```
pfe21% mkdir dir_name
pfe21% lfs setstripe -c 1 dir_name
pfe21% cd dir_name
pfe21% tar -xf tar_file
```

If there are large files in the same directory tree, it may be better to allow them to stripe across more than one OST. You can create a new directory with a larger stripe count and copy the larger files to that directory. Note that moving files into that directory with the mv command will not change the stripe count of the files. Files must be *created in* or *copied* to a directory to inherit the stripe count properties of a directory.

```
pfe21% mkdir dir_count_4
pfe21% lfs setstripe -c 4 dir_count_4
pfe21% cp file_count_1 dir_count_4
```

If you have a directory with many small files (less than 100 MB) and a few very large files (greater than 1 GB), then it may be better to create a new subdirectory with a larger stripe count. Store just the large files and create symbolic links to the large files using the symlink command ln.

```
pfe21%  mkdir dir_name
pfe21%  lfs setstripe -c 16 dir_name
pfe21%  ln dir_name/large_file  large_file
```

Increase the Stripe Count for Parallel Access to the Same File

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

The Lustre stripe count sets the number of OSTs the file will be written to. When multiple processes access blocks of data in the same large file in parallel, I/O performance may be improved by setting the stripe count to a larger value. However, if the stripe count is increased unnecessarily, the additional metadata overhead can degrade performance for small files.

By default, the stripe count is set to 4, which is a reasonable compromise for many workloads while still providing efficient metadata access (for example, to support the ls -l command). However, for large files, the stripe count should be increased to improve the aggregate I/O bandwidth by using more OSTs in parallel. In order to achieve load balance among the OSTs, we recommend using a value that is an integral factor of the number of processes performing the parallel I/O. For example, if your application has 64 processes performing the I/O, you could test performance with stripe counts of 8, 16, and 32.

TIP: To determine which number to start with, find the approximate square root of the size of the file in GB, and test performance with the stripe count set to the integral factor closest to that number. For example, for a file size of 300 GB the square root is approximately 17; if your application uses 64 processes, start performance testing with the stripe count set to 16.

Restripe Large Files

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

If you have other large files, make sure they are adequately striped. You can use a minimum of one stripe per 100 GB (one stripe per 10 GB is recommended), up to a maximum stripe count of 120. If you plan to use the file as job input, consider adjusting the stripe count based on the number of parallel processes, as described in the previous section.

If you have files larger than 15 TB, please contact User Services for more guidelines specific to your use case.

We recommend using the shiftc tool to restripe your files. For example:

1. Run ls -lh to view the size of the file(s):

   ```
   % ls -lh data/large_file data/huge_file
   -rw-rw---- 1 zsmith  g1001 555G Apr 14 22:21 data/large_file
   -rw-rw---- 1 zsmith  g1001 3.2T Apr 14 22:21 data/huge_file
   ```

   When a file is less than 1,200 GB, simply use one stripe per 10 GB. For a larger file, you can specify a maximum stripe count of 120.

2. Use shiftc to copy the file to a new file with this number of stripes:

   % shiftc --stripe=10g *large_file large_file*.restripe
   % shiftc --stripe=120 *huge_file huge_file*.restripe

3. Verify that the file was successfully copied. You should receive an email report generated by the shiftc command, or you can run shiftc --status. In the email or status output, check that the state of the operation is "done".

4. Move the new file in place of the old file:

   % mv *large_file*.restripe *large_file*
   % mv *huge_file*.restripe *huge_file*

For more information, see Using Shift for Local Transfers and Tar Operations.

Stripe Files When Moving Them to a Lustre Filesystem

Note: Skip this tip if you are using the PFL filesystems, nobackupp[*10-29*].

When you copy large files onto the Lustre filesystems, such as from Lou or from remote systems, be sure to use a sufficiently increased stripe count. You can do this before you create the files by using the lfs setstripe command, or you can transfer the files using the shiftc tool, which automatically stripes the files.

Note: Use shiftc (instead of tar) when you create or extract tar files on Lustre.

See the following articles for more information:

- Lustre Basics - Setting Stripe Parameters
- Shift Command Options

Reporting Problems

If you report performance problems with a Lustre filesystem, please be sure to include the time, hostname, PBS job number, name of the filesystem, and the path of the directory or file that you are trying to access.Your report will help us correlate issues with recorded performance data to determine the cause of efficiency problems.

Pleiades BeeGFS Filesystem

A solid state drive (SSD)-based BeeGFS filesystem, /nobackupp3, is in early stages of testing with a few users. Currently, it is only accessible from Pleiades front-end systems (PFEs) and compute nodes. The compute nodes on Electra or Aitken cannot access this filesystem.

The total space on /nobackupp3 is 192 terabytes (TB) with 16 stripes, each with a maximum of 12 TB. With its current configuration, you cannot set or change the stripe counts. Quota limits may be enabled in the future.

To view information about the filesystem, run:

```
% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf  --longnodes --spaceinfo
% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf --state --longnodes
```

WARNING: There is no failsafe mechanism for /nobackupp3. If one SSD fails, the data it contains cannot be recovered. Therefore, use Shift to copy any files you want to keep to your Lustre /nobackup filesystem (for temporary storage) or to Lou (for long-term storage). Be sure to properly stripe data copied to Lustre.

Using /nobackupp3

To use /nobackupp3 in a PBS job, you must ensure that it is mounted on all compute nodes before the job starts. To do so, include it as a needed site in your PBS script. For example, the following line mounts /nobackupp3, along with /home1 and /nobackupp8:

```
#PBS -l site=needed=/home1+/nobackupp3+/nobackupp8
```

Important: A job requesting Broadwell nodes might be run on Electra's Broadwell nodes instead of Pleiades. To run a job that uses Broadwell nodes and accesses /nobackupp3, you must specify Pleiades Broadwell nodes by using #PBS -l site=static_broadwell. For example:

```
#PBS -l select=xx:ncpus=28:mpiprocs=28:model=bro
#PBS -l site=static_broadwell
#PBS -l site=needed=/home1+/nobackupp3+/nobackupp8
```

For more information, see BeeGFS Basics.

A BeeGFS filesystem is a parallel cluster filesystem managed with the BeeGFS software. BeeGFS architecture is composed of four main services:

Management Service
: A very lightweight service for maintaining a list of all other BeeGFS services and their states.

Metadata Service
: Stores information about the data, such as directory information, file and directory ownership, and location of the user file contents on the storage targets. When a client opens a file, the metadata service provides the stripe pattern to the client. Metadata service is not involved in data read or write operations.

  There can be many server instances providing metadata services in a BeeGFS filesystem. Each service is responsible for its exclusive fraction of the global namespace. Having more metadata servers improves the overall system performance. Also, using faster processor cores for the metadata will have lower metadata access latency, providing better performance.

Storage Service
: Also referred to as the *object storage service*, this service stores striped user file contents.

  A storage server instance has one or more storage targets. The storage targets can be hard disk drives (HDDs) or the flash memory of solid state drives (SSDs).

  One strength of BeeGFS is its better performance compared to other filesystems in handling small I/O. This is because it automatically uses all available RAM on the storage servers for caching, so small I/O requests are aggregated into large blocks before the data is written to disk.

Client Service
: Mounts the filesystem to access the stored data. A BeeGFS client can be installed or updated without a system reboot.

Using the BeeGFS Command

The BeeGFS command `beegfs-ctl` can be used to perform administrative functions and show statistics:

```
pfe% which beegfs-ctl
/usr/bin/beegfs-ctl
```

To view `beegfs-ctl` options, run:

```
pfe% beegfs-ctl --help
```

BeeGFS Command Examples

To use these commands, you must specify a BeeGFS filesystem, using the `--cfgFile` option. In the examples, the Pleiades BeeGFS filesystem /nobackupp3 (`nbp3`) is specified.

- List the storage targets and their state:

  ```
  pfe% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf --state --longnodes
  ```

- Find the total, free and % free space of the storage targets:

  ```
  pfe% beegfs-ctl --listtargets --cfgFile=/etc/beegfs/beegfs-nbp3.conf  --longnodes --spaceinfo
  ```

- Find stripe pattern details of a directory or a file:

  ```
  pfe% beegfs-ctl --getentryinfo --cfgFile=/etc/beegfs/beegfs-nbp3.conf /path/to/file/or/directory
  ```

- List the node(s) used as metadata server(s):

  ```
  pfe% beegfs-ctl --listnodes --nodetype=meta --cfgFile=/etc/beegfs/beegfs-nbp3.conf
  ```

- List the nodes used as storage servers:

  ```
  pfe% beegfs-ctl --listnodes --nodetype=storage --cfgFile=/etc/beegfs/beegfs-nbp3.conf
  pfe% beegfs-ctl --listnodes --nodetype=storage --cfgFile=/etc/beegfs/beegfs-nbp3.conf --details
  ```

- List the storage pools:

  ```
  pfe% beegfs-ctl --liststoragepools --cfgFile=/etc/beegfs/beegfs-nbp3.conf
  ```

Additional Resources

- Pleiades BeeGFS Filesystem
- Introduction to BeeGFS (PDF)

Software

RECENT UPDATES: Due to NASA security requirements, the operating systems for nearly all HECC systems have been migrated from TOSS3 to TOSS4. The alternate operating environments :aoe=sles12 and :aoe=sles15 are no longer available on any of the compute nodes.

The production environment for most NAS systems is Red Hat Enterprise Linux-based Tri-Lab Operating System Stack (TOSS), developed at the U.S. Department of Energy.

This article provides some basic information to help you run applications on the TOSS4 operating system.

Pleiades Front Ends (PFEs) and the PFE Load Balancer

Using the Pleiades Front End Load Balancer will select pfe[*20-23*], which are still running TOSS3. For example:

```
your_local_host% ssh pfe
pfe23%
```

To find out the operating system of the PFE you are on, do:

```
pfe% grep . /etc/os-release
```

Compute Nodes

All cluster systems, including all CPU and CPU+GPU model types in Pleiades, Electra, Aitken, and Endeavour nodes run the TOSS image by default.

The following systems are currently being upgraded to Red Hat Enterprise Linux (RHEL8):

- The Lou front-end nodes, lfe[5-8].
- The Lou Data Analysis (LDAN) nodes.

Running PBS Jobs

Once the transition is complete, all jobs submitted to run on the compute nodes will use :aoe=toss4 by default.

PBS no longer schedules jobs requesting the previously available alternate operating environments :aoe=sles12 or :aoe=sles15.

Using Software Modules

On the PFEs and compute nodes booted with a TOSS 3 image, the default MODULEPATH includes the following:

```
/usr/share/modules/modulefiles
/nasa/modulefiles/toss3
/nasa/modulefiles/spack/gcc-4.8/
/nasa/modulefiles/pkgsrc/toss3/
```

On the Aitken Rome nodes booted with a TOSS4 image, the default MODULEPATH includes:

```
/usr/share/Modules/modulefiles
/nasa/modulefiles/toss4
/nasa/modulefiles/spack
/nasa/modulefiles/pkgsrc/toss4
```

Use the module avail command to find what software modules are added under these paths.

Building and Running Your Application

Many executables built under SLES 12 will also run under TOSS3. Similarly, executables built under TOSS3 may also run under TOSS4. Please check for both correctness and performance.

Some versions of HPE MPT are not compatible with TOSS4. We recommend using the default MPT version, which is accessed by specifying mpi-hpe/mpt as described in Using the NAS Recommended MPT Library. The default version will always be compatible with both TOSS3 and TOSS4.

Operating System

All NAS compute systems are running the Red Hat Enterprise Linux-based Tri-Lab Operating System Stack (TOSS).

To find the specific version on a host, run:

```
% grep . /etc/*release
```

To find the Linux kernel version number on a host, run:

```
% uname -r
```

Software on NAS Systems

Software programs on NAS systems are managed as modules or packages. Available programs are listed in tables below.

Note: The name of a software module or package may contain additional information, such as the vendor name, version number, or what compiler/library is used to build the software. For example:

- comp-intel/2020.4.304 - Intel Compiler version 2020.4.304
- mpi-hpe/mpt.2.25 - HPE MPI library version 2.25
- netcdf/4.4.1.1_mpt - NetCDF version 4.4.1.1, built with HPE MPT

Modules

Use the `module avail` command to see all available software modules. Run `module whatis` to view a short description of every module. For more information about a specific module, run `module help` *modulename*.

See Using Software Modules for more information.

Available Modules (as of December 20, 2021)

Module	**Function**
boost	C++ library
cdo	Climate and NWP data analysis tool
comp-aocc	AMD Optimizing C/C++ Compiler
comp-intel	Intel Compiler
cuda	NVIDIA parallel computing platform and programming model
dakota	Software toolkit from Sandia
fieldview	Data visualization & analysis tool
firefox	Web browser
gcc	C/C++ compiler
hdf4	I/O library & tools
hdf5	I/O library & tools
idl	Data visualization & analysis tool
mathematica	System & language for mathematical application
matlab	Numerical computing environment & programming language
metis	Graph, mesh, & hypergraph partitioning software
mpi-hpe	HPE MPI library
mpi-intel	Intel MPI library
ncl	NCAR Command Language
nco	NetCDF Operators
ncview	Visual browser for NetCDF files
netcdf	I/O library
nvhpc	NVIDIA HPC Software Development Kit
octave	Numerical computations language
openfoam	Open source CFD software
pagecache-management	Tool for limiting page cache usage
paraview	Data analysis and scientific visualization application
parmetis	Math/numerical library
petsc	Portable, Extensible Toolkit for Scientific Computation
pkgsrc	NetBSD Package Source collection of software packages
python3	Programming language (version 3)
scicon	Tools created by NAS Scicon team
savors	NAS in-house utility for synchronization and visualization of arbitrary streams
singularity	HPC container platform
szip	Compression library
tecplot	Data visualization & analysis tool
totalview	Debugger
vtune	Performance analysis tool
x2go	Open source remote desktop software for Linux
xxdiff	Graphical file & directories comparison & merge tool

Packages in the NetBSD Package Source Collection (pkgsrc)

Many software programs are now managed as packages on NAS systems, via the NetBSD Package Source Collection. To use a package, you must first access the current package collection by loading the pkgsrc module, which is listed in the Modules table.

See Using Software in the NetBSD Packages Collection for more information.

Partial list of NetBSD Packages in pgksrc/2021Q2

Module	Function
bioperl	Perl tools for computational molecular biology
bison	GNU yacc(1) replacement
blas	Basic Linear Algebra System
bmake	Portable (autoconf) bersion of NetBSD "make" utility
bzr	Bazaar open source distributed version control system
clang	C language family front-end for LLVM
cscope	Interactive C program browser
fetch	Client to fetch URLs
ffmpeg4	Decoding, encoding, & streaming software (v4.x)
fftw	Fast C routines to compute DFTs
gdal	Translator library for raster geospacial data formats
gettext	Tools for providing messages in different languages
git	GIT version control suite meta-package
GMT	Generic Mapping Tools
gnuplot	Portable, interactive function-plotting utility
grace	GRaphing, Advanced Computation & Exploration of data
GraphicsMagick	X application for displaying & manipulating images
h5utils	Utilities for conversion to/from HDF5
hdf5-c++	New generation HDF C++ wrappers
htop	Enhanced version of "top" utility
ImageMagick	Package for display & interactive image manipulation
lrzip	Long range ZIP or Lzma RZIP
lz4	Extremely fast compression algorithm
lzop	Fast file compressor similar to gzip, using the LZO library
mercurial	Fast, lightweight source control management system
metis	Unstructured graph partitioning & sparse matrix ordering system
mg	Small, fast, public domain Emacs-style editor
mono	Open-source implementation of .NET Development Framework
nano	Small, friendly text editor (a replacement for Pico)
ncview	Visual browser for netCDF format files
netcdf-fortran	Fortran support for NetCDF
nvi	Berkeley nvi with additional features
pbzip2	Parallel implementation of bzip2 block-sorting file compressor
pkgfind	Find packages by package name in pkgsrc
qgis	Geographic information system (GIS)
R	Statistical language for data analysis & graphics
rhash	Calculate/check CRC32, MD5, SHA1, GOST, or other hash sums
ruby	Ruby programming language
tkcvs	Tcl/Tk front-ends to CVS & diff
tmux	BSD-licensed terminal multiplexer (GNU screen alternative)
valgrind	Debugging & profiling tools
vtk	Visualization toolkit
wget	Retrieve files from the internet via HTTP & FTP

Software Directories

Software on the NAS systems include the operating systems, programming environments, and available software modules. You can find most of the software you need in the following directories:

Directory	Description
/nasa	Licensed or open-source software application modules
/PBS	Software for submitting, monitoring, and managing PBS jobs
/bin	Essential user command binaries, such as cp, ls, mv, vi
/lib	Essential shared libraries and kernel modules, such as libc, libm
/u/analytix/tools	Useful modules and environments provided by the NAS Data Analytics Group
/u/scicon/tools	Useful tools provided by the NAS Application Performance and Productivity Group
/usr/bin	Most user commands, such as cat, diff, ldd
/usr/lib	Libraries for programming and packages, such as libstdc++, libGL
/usr/include	General-use include files for the C programming language
/usr/local/bin	Binaries added for local use, such as acct_ytd, shiftc
/usr/local/lib	Shell startup files, such as glocal.cshrc for NAS systems

Except for those under /nasa, the binaries, libraries and include files listed above are included in your default search path.

NAS Software Policy

The NAS software policy was developed for the following reasons:

- NAS users have diverse and potentially conflicting software requirements.
- Revalidating code against new software components can be a time-consuming process that may not result in a demonstrable benefit in terms of performance, functionality, or reliability.
- Over time, maintaining an increasing number of software modules can make it difficult for users to find modules that meet their needs, and increases complexity for both users and support staff, as multiple versions will have different bugs and may depend on different versions of other software components.
- Outdated software versions may affect the reliability of code that depends upon the software.

Policy

The following policy applies to software modules installed on NAS systems and to any requests for the installation of new software:

- In general, no more than two versions of each software module are available.
- To provide a stable environment for users, software modules will remain available for at least one year (barring exceptional circumstances). After the first year, only the latest patch of each major/minor version will be retained. (For example, if v3.2.7 has been available for more than a year, and v3.2.9 is also available, v3.2.7 may be deprecated.)
- To request new software, contact NAS User Services. There must be sufficient demand before software is installed; otherwise, users will be expected to install the software themselves.
- Software that has been installed for more than three years and has not been used for more than one year may be deprecated and removed at the discretion of the system administrators.
- Commercial software that is no longer supported by the vendor may be deprecated and removed at the discretion of the system administrators.
- Deprecated software is available for only three months, and access is restricted. If you require access, please contact NAS User Services and provide a justification.
- Software that is no longer functional or presents an unacceptable security risk may be removed at any time and may bypass the deprecation and removal process.

Note: For software packages that contain multiple components, such as Perl and Python, the deprecation and removal process is based solely on the primary package version, without consideration of the additional components. For example, the Python software package includes additional components such as numpy and matplotlib. When a Python version is deprecated, the newer version might not include the same versions of these components.

Using Software Packages in pkgsrc

The NetBSD Packages Collection (pkgsrc) is a third-party software package management system for Unix-like operating systems. Among the over 17,000 packages available in pkgsrc, a few hundred of them have been built on Pleiades and are available in the public directory /nasa/pkgsrc.

A new collection of software packages is released every quarter by the pkgsrc developers. In addition to new packages or updated versions, each release is largely comprised of the same packages (with identical version numbers) as previous releases.

Software Packages Available on NAS Systems

For a list of software programs that are currently available as packages, see [Available Software Modules and Packages].

You can also see which pkgsrc modules are currently available by running module avail pkgsrc. For example:

```
pkgsrc/2020Q4  pkgsrc/2021Q2
```

are available on systems running TOSS 3, and

```
pkgsrc/2022Q1-rome
```

is currently the only one available on the Aitken Rome nodes running TOSS 4.

Note: On Endeavour3/4, which still uses the SUSE Linux Enterprise Server Version 12 (SLES 12) operating system, be aware that the only pkgsrc modules available are older than 2021.

Note: Although the discussion and examples below use pkgsrc/2021Q2 and the path /nasa/pkgsrc/toss3/2021Q2, you should use pkgsrc/2022Q1-rome and the path /nasa/pkgsrc/toss4/2022Q1-rome when using the Aitken Rome nodes.

All of the source distributions for building the pkgsrc modules are stored in the /nasa/pkgsrc/distfiles directory. If you plan to build software from source, check to see whether the source distribution is already there before downloading the files from the internet.

To view a complete list of all packages and a one-line description of each package, load a pkgsrc module and type the pkg_info command without any additional options. If you are looking for a particular package, you can check whether it is available by using the grep command plus a keyword from the output of the pkg_info command.

TIP: Because some package names contain capital letters, add -i to the grep command to ignore case distinctions.

For example:

```
pfe% module load pkgsrc/2021Q2
pfe% pkg_info | grep -i cairo
cairo-1.16.0nb4         Vector graphics library with cross-device output support
cairo-gobject-1.16.0nb5 Vector graphics library with cross-device output support
py39-cairo-1.20.1       Python bindings for cairo
```

You can also use the -e option to test the existence of a given package. For example:

```
pfe% pkg_info -e cairo

pfe% cairo-1.16.0nb4
```

Once you know the package name, you can use additional pkg_info options to find more information about the package. You can specify the full package name, or you can leave out the version number. For example:

```
pfe% pkg_info -B cairo-1.16.0nb4
```

or

```
pfe% pkg_info -B cairo
```

Some useful options for pkg_info include:

-B
: shows build information of the package

-d
: shows a long description of the package

-L
: shows the files within the package

-N
: shows what other packages were used to build the package

-R
: shows what other packages require the package

Using Packages

The root directory of a pkgsrc release (for example, /nasa/pkgsrc/toss3/2021Q2) contains familiar directories such as bin, sbin, lib, include, man, and info. For most packages, you can find their executables, libraries, or include files in these directories.

Using an Executable

When you load a pkgsrc module, the paths to its bin and sbin directories are prepended to the PATH environment variable, as shown in the following example. Therefore, you do not need to provide full paths to these directories in order to run executables in them.

```
pfe% module show pkgsrc/2021Q2
-------------------------------------------------------------------
/nasa/modulefiles/pkgsrc/toss3/pkgsrc/2021Q2:

system           /bin/logger -p local2.info -t envmodule ychang1 display pkgsrc/2021Q2
module-whatis    A collection of software built via NetBSD's pkgsrc
prepend-path     INFOPATH /nasa/pkgsrc/toss3/2021Q2/info
prepend-path     MANPATH /nasa/pkgsrc/toss3/2021Q2/man
prepend-path     PATH /nasa/pkgsrc/toss3/2021Q2/bin:/nasa/pkgsrc/toss3/2021Q2/sbin
setenv           PKGSRC_BASE /nasa/pkgsrc/toss3/2021Q2
-------------------------------------------------------------------
```

Linking a Library

Unlike the PATH environment variable, loading a pkgsrc module does not set the LD_LIBRARY_PATH environment variable. The reason for this is to prevent libraries in the pkgsrc directory from overriding those with the same filenames at the system default locations or directories that may have been included in your existing LD_LIBRARY_PATH.

To link a static library (for example, libcairo.a) in the /nasa/pkgsrc/toss3/2021Q2/lib directory:

```
-I/nasa/pkgsrc/toss3/2021Q2/include
-L/nasa/pkgsrc/toss3/2021Q2/lib -lcairo
```

To link to a dynamic library (for example, libfftw.so) in the /nasa/pkgsrc/toss3/2021Q2/lib directory, follow the recommendations in the list below, in order of preference.

Note: Dynamic libraries have the suffix ".so" rather than the suffix ".a", which is used for static libraries.

- Set the LD_RUN_PATH environment variable as shown below prior to the link step. This will get /nasa/pkgsrc/toss3/2021Q2/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  ```
  setenv LD_RUN_PATH /nasa/pkgsrc/toss3/2021Q2/lib
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  ```

- Add the -Wl,-R linker option during the link step. This will also get /nasa/pkgsrc/toss3/2021Q2/lib encoded into the executable. Setting LD_RUN_PATH or LD_LIBRARY_PATH is not needed during run time.

  Note: The linker option -Wl,-R takes precedence over the LD_RUN_PATH setting.

  ```
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  -Wl,-R/nasa/pkgsrc/toss3/2021Q2/lib
  ```

- Do not set LD_RUN_PATH or use the -Wl,-R linker option during the link step. Since /nasa/pkgsrc/toss3/2021Q2/lib is not encoded into the executable, you must set LD_LIBRARY_PATH during run time.

  Link time:

  ```
  -I/nasa/pkgsrc/toss3/2021Q2/include
  -L/nasa/pkgsrc/toss3/2021Q2/lib -lfftw
  ```

  Run time:

  ```
  setenv LD_LIBRARY_PATH "/nasa/pkgsrc/toss3/2021Q2/lib:$LD_LIBRARY_PATH"
  ```

For more information about using the setenv command, see **man setenv**.

The files of some packages are not installed in the bin, lib, and include directories. For example, the guile package is installed in the /nasa/pkgsrc/toss3/2021Q2/guile directory, which has its own bin, lib, and include subdirectories. The best way to find the exact location of a file you need from a package is to run the pkg_info command as follows:

```
pfe% pkg_info -L package_name
```

Separate modules have been created for some commonly used packages to make them easier to use—you can simply load the modulefile without loading a pkgsrc module. On systems running TOSS 3, these include:

- boost/1.76
- gcc/7.5
- gcc/9.3
- gcc/10.2
- gcc/10.3
- python3/3.9.5

- octave/6.2

On the Aitken Rome nodes running TOSS 4, the following modules from pkgsrc have been made available:

- boost/1.78
- gcc/6.5
- gcc/7.5
- gcc/8.4
- gcc/9.3
- gcc/10.3
- python3/3.9.12
- octave/6.4

Note: /usr/bin/python will no longer be linked to python2. To use python2, you must specify the whole path /usr/bin/python2.

Separate modules for other packages can be created when there is demand from the NAS user community.

For more information about pkgsrc, see https://www.pkgsrc.org.

Using virtualenv to Manage Your Own Python Environment

NAS provides several Python environments including the system default, /usr/bin/python3 (currently version 3.6.8), and modules such as python3/3.9.5, and so on. These environments provide a variety of Python packages and versions. However, if you need a Python package or version that is not available in these environments, we recommend that you install it in your own directory.

To install a Python package or version, we recommend using the virtualenv tool, which provides these benefits:

- It creates an isolated Python virtual environment. When activated, you can install and use packages through this environment, instead of through /usr/bin/python or the NAS-provided Python modules.
- You can create separate virtual environments for separate projects that may need different versions of the same package.

Complete these steps to check whether a package is available on NAS systems, create a virtual environment, install a package from the Python Package Index (PyPI), and use the package in this virtual environment:

1. Load the latest Python module (currently python3/3.9.5), which already has virtualenv, pip and many other Python packages installed:

   ```
   % module load python3/3.9.5
   ```

2. View the packages and versions that are available under python3/3.9.5:

   ```
   % pip list
   ```

   In the output, look for a package with the version you want. If it is not available, or if you want to install your own private copy, continue with the steps below.

3. Create a virtual environment with a new directory name, such as ~/myenv. The virtualenv tool will create the directory for you.

   If you want this virtual environment to be able to access packages already available in the global site-packages (in this example, /nasa/pkgsrc/toss3/2021Q2/lib/python3.9/site-packages), so that you do not have to install everything from scratch, run:

   ```
   % virtualenv --system-site-packages ~/myenv
   ```

   Otherwise, run:

   ```
   % virtualenv ~/myenv
   ```

   At the end of this step, you should see the python (which is linked to python3.9), pip (which is same as pip3 or pip3.9), wheel, and multiple activate scripts under ~/myenv/bin.

4. Activate the virtual environment:

   ```
   % source ~/myenv/bin/activate (for bash)
   [myenv]%

   or

   % source ~/myenv/bin/activate.csh (for csh)
   [myenv]%
   ```

   Your prompt should now include the virtual environment [myenv].

5. Use the pip tool from this environment to install or upgrade a package (for example, SymPy) under ~/myenv/lib/python3.9/site-packages, and run the pip list command to check whether the installation is completed properly. (Note that SymPy depends on mpmath, so make sure that mpmath is installed before installing sympy.)

   ```
   [myenv]% which pip
   /u/username/myenv/bin/pip
   [myenv]% pip install --upgrade sympy
   [myenv]% pip list
   ```

6. When you finish using this isolated environment, run the deactivate command to exit. After you exit, your prompt should no longer include [myenv].

   ```
   [myenv]% deactivate
   %
   ```

To install another package or to use a package in this isolated virtual environment at a later time, you do not need to load the python/3.9.5 module again. Simply activate the virtual environment (step 4, above), then use the packages or install additional packages. For example:

```
% source ~/myenv/bin/activate (for bash)

or

% source ~/myenv/bin/activate.csh (for csh)

[myenv]% pip install --upgrade pendulum
```

```
[myenv]% pip list
[myenv]% which python
/u/username/myenv/bin/python
[myenv]% python
>>> import sympy as s
>>> import pendulum as p
```

Using the Intel Distribution for Python

Intel Distribution for Python provides accelerated performance for numerical computing and data science on Intel architectures. With the distribution, math and statistical packages such as NumPy, SciPy, and scikit-learn are linked with Intel's performance libraries for near-native code speeds. Libraries include:

- Math Kernel Library (Intel MKL) with optimized BLAS, LAPACK, FFT, and random number generators
- Message Passing Interface (Intel MPI)
- Thread Building Blocks (Intel TBB)
- Data Analytics Acceleration Library (Intel DAAL)

In general, you do not need to change your Python code to take advantage of the improved performance Intel's Python Distribution provides. However, for random number generators, we recommend using the MKL-based random number generator numpy.random_intel as a drop-in replacement for numpy.random.

For machine learning, Intel Distribution for Python provides deep learning software such as Caffe and Theano, as well as classic machine learning libraries, such as scikit-learn and pyDAAL (which implements Python bindings to Intel DAAL).

Accessing Intel Python on Pleiades

On Pleiades, you can access a full standalone version of Intel Distribution for Python via a module in the /nasa directory.

```
pfe% module load comp-intel/2018.3.222
pfe% module load python3/Intel_Python_3.6_2018.3.222
pfe% which python
/u/scicon/tools/opt/sles12/python/2018.3.222/intelpython3/bin/python
pfe% python --version
Python 3.6.3 :: Intel Corporation
```

Note: Because the Python 2 end-of-life date is 2020, only the Intel Python 3 distribution is made available on Pleiades to encourage Python 2 users to migrate to Python 3.

To see what packages are included in this distribution, run:

```
pfe% conda list
```

WARNING: Do not load any other modules if you don't need them in your working session, as there are potential software conflicts between the Intel Python and other modules such as Tecplot, Python from other distributions, and various MPI libraries.

If You Need Additional Packages

Only NAS system administrators can add packages to the /nasa or /u/scicon directory, where this Intel Python module is installed. If you need additional packages that are not currently included, and you think the packages might be widely used by other Pleiades users, send a request to support@nas.nasa.gov. Otherwise, follow the instructions below to add packages to your own directories.

Installing Packages from the Python Package Index

If the package is available at The Python Package Index (PyPI), use pip to install it in your $HOME/.local/lib/python3.6/site-packages directory, where Intel Python will be able to find the package.

To install a package, run:

```
pfe% which pip
/u/scicon/tools/opt/sles12/python/2018.3.222/intelpython3/bin/pip
pfe% pip install --user package_name
```

Note: The virtualenv tool for managing Python environments is not available in the Intel Distribution for Python.

Installing Packages from the Anaconda Repositories

If the package is available at Anaconda Repositories, use the conda tool to install it.

Note: Unlike pip install, there is no --user option for conda install. Instead, you must create a new Conda enviroment where all the packages you need are available, including the Intel Distribution for Python.

Use one of the following two methods to create a new Conda environment and install packages.

Method 1: Clone the Intel Python from the /u/scicon Directory

Clone the entire Intel Python Distribution from /u/scicon to your $HOME/.conda/envs directory and add the packages to the new environment:

```
pfe% conda create -n my_clone_env --clone="/u/scicon/tools/opt/sles12/python/2018.3.222/intelpython3"
pfe% conda install -n my_clone_env package_name
```

Note: Cloning the entire distribution requires ~3.6 GB of space in your $HOME directory.

When you want to use the *my_clone_env* environment, you will need to activate that environment under bash, because csh is not supported by conda. Remember to deactivate when you are done using that environment.

```
pfe% bash
pfe% source activate my_clone_env
(my_clone_env) bash$ which python
/homeX/username/.conda/envs/my_clone_env/bin/python
(my_clone_env) bash$ #do your work
(my_clone_env) bash$ source deactivate
```

Method 2: Install Your Own Anaconda or Miniconda Distribution

Instead of using the Intel Python provided in the /u/scicon directory, you can install your own Anaconda or Miniconda distribution with the packages you want, and follow the steps in the document Installing Intel® Distribution for Python and Intel Performance Libraries with Anaconda to add some of the packages from the distribution.

To learn more about using conda, see conda -h or download the Conda Cheat Sheet.

Additional Resources

- Intel Distribution for Python Documentation
- Intel Distribution fo Python - Highlights & Overview
- Intel Distribution for Python - Build for Speed
- Techniques Used to Accelerate Performance of numpy and scipy in Intel Distribution for Python
- Achieving High-Performance Computing with the Intel Distribution for Python

Using Intel MKL-Optimized Python Packages on Pleiades and Electra

Several packages in the Intel Distribution for Python are optimized for use with the Intel Math Kernel Library (MKL), including:

caffe	distarray	h5py	matplotlib	numba
numexpr	numpy	opencv	pandas	pydaal
pytables	pywavelets	scikit-image	scikit-learn	scipy
tensorflow	theano			

Note: The packages are categorized into a full list or a core list, as described in the Complete List of Packages for the Intel Distribution for Python. Some packages that are not in the core list, such as caffe and distarry, are not included in the NAS-provided Intel Python module. If there is sufficient demand for these packages, they may be added to the module.

You do not need to make any changes to your Python application in order to use these packages. However, to get the best possible performance on Intel Xeon multi-core processors, it is important that your application employs multi-threading to achieve effective multi-core parallelism.

You can experiment with different environment variable settings to achieve the best performance for your application, as shown in the following example.

Performance Tuning Example

In this example, more than 3x speedup was obtained for the convolutional.py code, which uses TensorFlow. The code was run on Electra using this PBS script:

```
##### PBS script ######
#PBS -lselect=1:ncpus=40:model=sky_ele
module load comp-intel/2018.3.222
module load python3/Intel_Python_3.6_2018.3.222

python3 convolutional.py
########################
```

The following wall times were obtained for convolutional.py by using different environmental variable settings:

- Wall time: 00:09:27

  The NAS default value of 1 for OMP_NUM_THREADS was not changed.
- Wall time: 00:03:19

  The value of OMP_NUM_THREADS was changed to 20, as follows:

  ```
  setenv OMP_NUM_THREADS 20
  ```
- Wall time: 00:02:53

  Four environment variables were set as follows:

  ```
  setenv OMP_NUM_THREADS 20
  setenv KMP_BLOCKTIME 30
  setenv KMP_AFFINITY "granularity=fine,verbose,compact,1,0"
  setenv KMP_SETTINGS 1
  ```

Each of these variables is described in the next section.

Recommended Environment Variables for Tuning Performance

You can use the following OpenMP (OMP_) and Intel Extensions (KMP_) environment variables to help tune multi-threading performance:

OMP_NUM_THREADS
: Sets the maximum number of threads to use for OpenMP parallel regions if no other value is specified in the application.
: **Default set by Intel:** The number of processors visible to the operating system
: **Default set by NAS:** 1 for PBS jobs at NAS to avoid oversubscription of resources

KMP_BLOCKTIME
: Sets the time (in milliseconds) that a thread should wait, after completing the execution of a parallel region, before sleeping.
: **Default:** 200

KMP_AFFINITY
: Enables runtime library to bind threads to physical processing units.
: **Default:** noverbose,warnings,respect,granularity=core,duplicates,none
: **See also:** Using Intel OpenMP Thread Affinity for Pinning

KMP_SETTINGS
: Enables (true) or disables (false) the printing of OpenMP runtime library environment variables during program execution. Two lists of variables are printed: user-defined environment variables settings, and effective values of variables used by OpenMP

runtime library.
Default: false

Additional Resources

- Intel Distribution for Python
- Composable Multi-Threading and Multi-Processing for Numeric Libraries
- Large Matrix Operations with SciPy and NumPy: Tips and Best Practices
- TensorFlow Performance Guide
- TensorFlow Optimizations for the Intel Xeon Scalable Processor
- TensorFlow Optimizations on Modern Intel Architecture

Installing R Packages in Your Own Directories

R is a language and environment for statistical computing and graphics that is available through the NetBSD Packages Collection (pkgsrc) on Pleiades. R can easily be extended with packages written by others in the R community. The pkgsrc build on Pleiades includes the basic R package and a few other R packages, such as the Interface to Unidata netCDF data files (ncdf) and Regression Spline Functions and Classes (splines).

You can install additional R packages for your own use. By default, the system attempts to install the packages in the /lib/R/library subdirectory under the pgksrc root directory where you do not have write permission. Instead, you can install R packages in your own directories.

Recommendation: Pre-create a directory (such as ~/R-packages) and set the R_LIBS environment variable before installing R packages. (See sample steps below.)

Alternatively, you can use one of these methods to install R packages in your own directory:

- Pre-create a directory (such as ~/R-packages). During installation, add the lib="/u/your_username/R-packages" option to the install.packages function.
- During the installation step that includes the install.package function, answer yes (y) to both of these questions: (1) Would you like to use a personal library instead? and (2) Would you like to create a personal library ~/R/x86_64-unknown-linux-gnu-library/?r-version-number?

In the following sample steps, the recommended method described above is used to install the xts package (along with the zoo package, on which it has a dependency). The example shows a few possible arguments you can choose for the install.packages function.

1. Pre-create a directory and set the R_LIBS environment variable:

   ```
   pfe27% mkdir ~/R-packages
   pfe27% setenv R_LIBS ~/R-packages (for csh)
   or
   pfe27% export R_LIBS=~/R-packages (for bash)
   ```

2. Load the pkgsrc module and start R:

   ```
   pfe27% module load pkgsrc/2021Q2
   pfe27% R
   ```

3. At the R prompt >, choose one of the following three methods to identify a download site for the packages from the Comprehensive R Archive Network (CRAN) and install the packages:

   ```
   > install.packages("xts")
   ```

   If you use this method, a window pops up with a list of HTTPS CRAN mirror sites. Scroll to the end of the list and click(HTTP mirrors). A second page will open showing multiple HTTP mirror sites. Choose one of these HTTP sites as a source for downloading the zoo and xts packages.

   ```
   > install.packages("xts",repos="http://cran.r-project.org")
   ```

   You can specify a different site, such as http://cran.cnr.berkeley.edu.

   ```
   > install.packages("http://cran.r-project.org/src/contrib/zoo_1.8-0.tar.gz", type="source")
   > install.packages("http://cran.r-project.org/src/contrib/xts_0.9-7.tar.gz", type="source")
   ```

For more information about R, including technical documentation, see the R Project website.

How to Run a Graphics Application in a PBS Batch Job

In order to run a graphics application in a PBS batch job, you must provide a value for the **DISPLAY** environment variable—even if you do not have or need an actual display for the job. If the batch job cannot find this value, it will quit and return an error such as "Can't open display" or "Display not set."

To provide a value for the **DISPLAY** variable, you can set up a display connection using either X11 forwarding or a VNC session, or you can run the job on the Pleiades GPU nodes, which include graphics cards. Each of these methods is described below.

Method 1: Use X11 Forwarding to Set Up a Display

Complete these steps to use your local system as an remote X11 server, which acts as the display:

1. Set up X11 Forwarding (if it's not set up already).
2. Log in to a Pleiades front end (PFE) and submit a PBS batch job with the -V option on the qsub command line or inside the PBS script. For example:

   ```
   your_local_system% ssh pfe23
   pfe23% echo $DISPLAY
   10.150.27.21:30.0
   pfe% qsub -V job_script
   ```

   In the above example, using the -V option will export the **DISPLAY** environment variable (with its value set to 10.150.27.21:30.0) from your PFE login session to the PBS batch job.

Drawbacks to using this method:

- Because the connection from the PBS job to your local system must be maintained, you cannot exit from either your local system or from the PFE login window until the job completes.
- Application performance on remote X11 servers deteriorates with latency, so unless your local system is physically close to Pleiades, job performance may not be optimal.

Method 2: Use a VNC Session to Set Up a Display

Complete these steps on a PFE to set up a VNC viewer, which acts as the display:

1. Complete steps 1-4 of the instructions in VNC: A Faster Alternative to X11 to establish a VNC session.

   Once the VNC viewer is set up and running, you will see the following messages (or something similar) in the PFE login window:

   ```
   pfe23% vncserver

   New 'X' desktop is pfe23:9

   Starting applications specified in /u/username/.vnc/xstartup
   Log file is /u/username/.vnc/pfe23:9.log
   ```

   This information indicates that the **DISPLAY** value pfe23:9.0 is set, providing a valid display that your PBS batch jobs can access.

2. In your PBS batch job script, add the following line to instruct the job to use this display:
   - For csh: setenv DISPLAY pfe23:9.0
   - For bash: export DISPLAY=pfe23:9.0
3. Submit your PBS batch job from the PFE connection window where the VNC session is running.

Benefits of using this method:

- Job performance is not impacted.
- The VNC viewer will be maintained as long as the PFE you started it on (pfe23 in this example) is online. The session will stay open even if you exit from your local system or from the PFE connection window where the viewer is running. To reconnect to the session, simply log in to the PFE and repeat steps 3 and 4 in the VNC instructions with the same port number (5909 in this example).

Note: The number of VNC ports on each PFE is limited. You must stop the vncserver script after your PBS job completes, or the port will remain open and owned by you, which prevents other users from using it. Remember to complete step 5 in the VNC instructions to properly shut down your VNC session.

Method 3: Run Your Job on the Pleiades GPU Nodes

When you run a job on a GPU node, a display is enabled automatically.

Complete these steps to run your job on a GPU node:

1. Modify your PBS script to run startx and set **DISPLAY** to :0.0 by adding these lines:

```
/bin/startx /usr/bin/Xvnc :1 -s 0 -ac -br -listen tcp passwordFile=$HOME/.vnc/passwd &
setenv DISPLAY :1.0       (for csh)
or
export DISPLAY=:1.0       (for bash)
sleep 15  # wait 15 seconds to allow the startx to fully start up before using the display
```

2. Submit a batch job to request a GPU node using the correct queue:

   To request a GPU node:

   ```
   pfe23% qsub -q v100 -lselect=1:model=sky_gpu pbs_script
   ```

Drawbacks to using this method:

- There are fewer GPU nodes available than non-GPU nodes.
- You cannot use this method if your graphics application needs more than 64 GB of memory.

Virtual Network Computing (VNC) software provides a way to reduce X11 overhead on high-latency networks such as the Internet. In practical terms, once a VNC session is underway, latencies are on the order of seconds rather than minutes. VNC can make remote X11 applications useful instead of being tedious and non-productive.

The principle of operation involves a host server process (for example, Xvnc) that communicates with X11 applications running on Pleiades. The host server process transmits images and image updates using a low-overhead protocol to the remote system's viewer client.

Security and Firewalls

In the NAS environment VNC traffic is carried by a SSH tunnel, similar to the way SSH is used to tunnel X11 traffic. Using an SSH tunnel provides security because SSH encrypts tunnel traffic in both directions. If you are already using SSH, then VNC traffic will travel to/from NAS systems over current connections and through current firewalls. There is no need for any additional communication updates or authorizations.

Where is the VNC Software?

The Pleiades system runs on Linux. All of the necessary VNC software is installed in /usr/bin.

You do not need to run an X11 server on the remote system (your local system) because in the VNC environment, all of the X11 work is done on the Pleiades front-end systems (pfe[*20-27*]). However, you do need a VNC client viewer. The client might already be installed in many Linux distributions and on recent versions of Mac OS X; if it is not installed on your system, you will need to download the client.

If you have a NAS-supported system, please note that:

- For NAS-supported Linux workstations, a VNC client viewer (RealVNC version 4.1.2) should be installed in /usr/bin/vncviewer.
- For NAS-supported Mac workstations, you can download a VNC client called TurboVNC Viewer from the miscellaneous category in the Self Service app. Self Service can be found through Spotlight (the magnifying glass in your system's top navigation bar) or under the Applications folder.

If you have a Windows desktop system, you can download free VNC clients from the following websites:

- Real VNC
- Tight VNC
- UVNC
- TigerVNC

Ask your local system administrator for help to install the VNC client software.

Steps to Establish a VNC Session

In the following steps, pfe24 is used as an example; you can substitute another PFE.

Note: Although there are other ways to establish a VNC session, this method is convenient as it does not require you to manually find an available display number to use.

Before You Begin

VNC is much easier to use if you set up SSH Passthrough on your local system. In your .ssh/config file on your local system, you do not need to enable SSH X11 forwarding, but you must include the line ForwardAgent yes.

Known Issue: Make sure you do not have a MATLAB, Tecplot, or FieldView module loaded when you invoke vncserver. Once the VNC session is established, you can load the module.

Step 1: Connect to the PFE

Once SSH Passthrough is set up properly, you can establish a SSH connection from your local system to pfe24:

```
your_local_system% ssh pfe24
pfe24%
```

Step 2: Run the vncserver Command on pfe24

vncserver is a script that starts/stops/kills the actual VNC server, Xvnc.

The first time you invoke vncserver on a server, you will be prompted to create a password for VNC that is up to 8 characters in length. (If you create a longer password, it will be truncated to 8 characters.) This password is encrypted and saved in the $HOME/.vnc/passwd file on the server. Once this is done, you will not be prompted for a password on the server when you invoke vncserver for subsequent VNC connections.

Run vncserver as follows:

```
pfe24% vncserver -localhost

You will require a password to access your desktops.

Password:  <--- type in a password of your choice

Warning: password truncated to the length of 8.
Verify:  <-- retype your password

New 'X' desktop is pfe24:25

Creating default startup script /u/username/.vnc/xstartup
Starting applications specified in /u/username/.vnc/xstartup
Log file is /u/username/.vnc/pfe24:25.log
```

There are a few options to the vncserver command, such as :display (for setting the display number), -geometry (for setting the desktop width and height in pixel), etc. The -localhost option shown in the above example is a local security option that you should use all the time. It must appear as the last option or it won't get processed.

Similar to an X11 session, a VNC session uses a display number. If not supplied, the vncserver searches over the valid range from 0 to 99 and assigns the next free display number for your session. In the above example, a display number of 25 is assigned.

Step 3: Create a SSH Tunnel from Your Local System to the Server

The next step is to create a SSH tunnel from your local system to the server. This is done by first escaping into an SSH sub-shell and specifying a local client's port number and a server's port number to use. The default SSH escape characters are ~C (upper case 'C'). If you do not get the SSH prompt, repeat the ~C.

```
pfe24% ~C
ssh> -L 59xx:localhost:59xx
Forwarding port.
```

At the SSH prompt, provide a local client port and a remote server port. VNC by default uses TCP port 5900+xx. Thus, it is common to provide the value 59xx for both the local client port (the number before localhost) and server port (the number after localhost). The value for *xx* is obtained from the final output from the vncserver startup command. In the example shown in Step 2, a vncserver was started on pfe24:25, so in this scenario *xx* would have a value of 25. The port number would therefore be 5925.

Note that the client port number and the server port number do not need to be the same. Some may suggest using a very high client port number such as 22222 or 33333 since high port numbers are less likely to be reserved for other purposes. For example:

```
pfe24% ~C
ssh> -L 22222:localhost:5925
Forwarding port.
```

The maximum number allowed for the client port is 65535. Avoid using the local port numbers 0-1024 (root privilege required), 5900 (for Mac systems, reserved for some Apple remote desktop products), and 6000-6063 (reserved for local X window server). Use the netstat -an command to check what local port numbers have been used:

```
your_local_system% netstat -an | less
tcp46      0      0  *.5900                *.*                    LISTEN
tcp4       0      0  *.22                  *.*                    LISTEN
```

The above example shows local ports 5900 and 22 are in use and should be avoided.

Step 4: Start the VNC Viewer Application on Your Local System

- If your local system is a Mac and you have downloaded TurboVNC Viewer, launch it. For the VNC server, enter localhost:*display number* and click **Connect**.

  In the popup window, enter your VNC password in the **Password** field.

- If your local system is a Linux system, run:

  ```
  your_local_system% vncviewer localhost:localportnumber
  ```

  You should get a password prompt. Enter your VNC password that you created on the server.

  The *localportnumber* is the one you use in step 3. For example, if you choose 22222 as your local port, run:

  ```
  your_local_system% vncviewer localhost:22222
  ```

If everything goes well, the Xvnc server will send a X11 window manager display to your local system that will appear as an xterm in the viewer's window.

The default window manager is TWM, and there are a couple other window managers to choose from in the /usr/bin directory, including FVWM, MWM, IceWM, and GNOME. The GNOME window manager provides a GUI view of a user's files and includes a few useful tools.

To use a non-default manager, modify your $HOME/.vnc/xstartup file on the host where your start vncserver. For example:

```
#twm &
/usr/bin/gnome-session
```

You can also change the size and position of the xterm in your viewer's desktop by changing the values in the following line of the $HOME/.vnc/xstartup file on the host where you start vncserver. For example:

```
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
```

This specifies an xterm that is 80 characters wide, 24 characters high, at a position (10 pixels, 10 pixels) from the upper left corner of the VNC viewer's desktop.

TIP: The modifications to the xstartup file only take effect for a new VNC connection. You will need to stop the existing VNC server and start a new one.

The window manager's xterm is running on pfe24 itself. From this xterm, you can do tasks that you normally do on pfe24—for example, start an X application or ssh to other NAS systems. PBS jobs can also connect to a VNC session. Specifically, in the xterm in the viewer's window, submit an interactive PBS job with the -X option (upper case 'X') and do not reset the DISPLAY variable before starting an X application:

```
pfe24% qsub -I -X -lselect=1:ncpus=28:model=bro,walltime=1:00:00
qsub: job 1030046.pbspl1.nas.nasa.gov ready
PBS> xclock
```

TIP: Your VNC session and the interactive PBS job will continue to be active even if you disconnect from the Pleiades front end where you started vncserver. Assuming the PFE where you started vncserver is not down, you can reconnect to the same VNC session: simply ssh into the PFE (pfe24 in this example) and repeat steps 3 and 4 with the same port number that you used before (5925 in this example). If the interactive PBS session has not reached its wall time limit, the PBS job will be there waiting.

Step 5: Shut Down the Server When You are Done with the VNC Session

On each VNC server, there are a limited number of VNC sockets available. At the end of a session, be sure to exit the VNC application on your local system so that others can use the sockets. In the terminal window where you started up VNC, use the following command to clean up a few temporary socket files vncserver had created.

```
pfe24% vncserver -kill :xx (supply the original display number)
```

For example:

```
pfe24% vncserver -kill :25
Killing Xvnc process ID 3435054
```

WARNING: Don't manually kill vncserver. Doing so will leave lock and socket files (for example, /tmp/.X11-unix/X25, $HOME/.vnc/pfe24:25.pid, etc.) on the server.

TIP: If you get a black screen on your VNC viewer, try the following methods to resolve the issue:

1. Check /tmp/.X11-unix for any existing VNC sessions, and clean them up by using the vncserver -kill :xx command, as described in Step 5 above.
2. If you normally load MATLAB, Tecplot, or other GUI application modules, unload them before you start vncserver. The LIBGL_ALWAYS_INDIRECT=y setting in these modules is known to cause the black screen.
3. If unloading the MATLAB and Tecplot modules does not solve the problem, use the twm window manager instead of icewm or gnome-session in your .vnc/xstartup file. For unknown reasons, it is possible that after you resolve the black screen issue by using twm, you can revert back to using other window managers.

See the commands described below to learn how to use modules.

Note: Software programs on NAS systems are managed either as modules or packages. For information about using software programs that are managed as packages, see Using Software in the NetBSD Packages Collection.

Using Modules

To find out what software modules are available on NAS systems, run the `module avail` command. You can also view a short description of every module by running `module whatis`, or find information about a specific module by running `module help` *modulename*.

To use the software provided in a module, you must first load the module by running the `module load` command. You can load several modules at the same time, as follows:

```
% module load module_name_1 module_name_2 ... module_name_N
```

Testing Modules

New software modules and new versions of existing modules are sometimes available for testing before they are released for general use. To access the test modules, run:

```
module use -a /nasa/modulefiles/testing
module avail
module load module_name
```

If you use the test modules, please send your feedback to support@nas.nasa.gov. Let us know whether the modules worked well or, if you had any problems, describe any steps you took to reproduce them.

Note: Test modules may be moved into production or deleted without notice.

Additional Module Commands

Some useful module commands include:

module list
: Lists which modules are already loaded in your environment.

module purge
: Removes all the modules that are loaded in your environment.

module switch *old_module_name new_module_name*
: Enables you to switch between two modules.

module show *module_name*
: Shows changes that will occur in your environment if you load *module_name.*

For more information, see **man module**.

TIP: If the module commands don't work, ensure that the following line
is included in your .cshrc or .profile file:

- For csh: `source /usr/local/lib/global.cshrc`
- For bash: `source /usr/local/lib/global.profile`

Creating Your Own Modulefiles

If you maintain software for your own use or to share with your group, it may be useful to create modulefiles that can be loaded, purged, and so on, just like any other modulefile.

To create your own modulefile, you can simply copy an existing modulefile to your directory (commonly named /u/*your_username*/privatemodules), and edit it as appropriate.

First, find a module that you want to mimic and run `module show` *module_name*. The modulefile itself will be listed at the top of the `module show` output. You can find most available modulefiles in the /nasa/modulefiles/toss3 directory.

Here are some common features in well-designed modulefiles:

```
#%Module
##
##  All modulefiles must begin on the first line with those 8 characters
##  Here, in the comments, you can describe how the package was built, or, better yet,
##  put it in the ModulesHelp section below

proc ModulesHelp { } {
  puts stderr "Put the information you want to display when the user
```

```
   does module help on this"
   puts stderr "modulefile. Note each continuation line begins with puts
   stderr and comments in double quotes"
}

# include the following line only if you want to log the loading of the
# modulefile in our system logs
# most users would leave this out
source /nasa/modulefiles/common/log_module_usage.tcl

# if there might be several versions of this software, it would be good to
#set the version number
set version 1.0-something

# setting basepath or SWDIR is for naming the root directory where your
# various bin, include, lib directories reside
# see later in the modulefile on how it is used
set basepath /u/your_userid/dir_name/dir_name

# if this software requires other modules to be loaded first,
# then use the prereq specifier, for example:
prereq comp-intel

# if your software requires other software that you build,
#then you might simply load them, as in:
module load  my_other_module1  my_other_module2

# if this software conflicts with another software that the user
# may inadvertently load, then use:
conflict another_modulefile

# Finally, the actual setting of the PATHs etc.  You have a choice of either
# prepend-path or append-path:
prepend-path   PATH                 $basepath/$version/bin
prepend-path   CPATH                $basepath/$version/include
prepend-path   FPATH                $basepath/$version/include
prepend-path   LD_LIBRARY_PATH      $basepath/$version/lib
prepend-path   LIBRARY_PATH         $basepath/$version/lib
prepend-path   MANPATH              $basepath/$version/share/man
```

If you plan to share this software with other users in your group, make sure that they have read permission to every file and directory, and execute permission on all directories and the executables in the software's bin directory.

In order to see your newly created modulefile in the output of the module avail command, run the module use command first, as follows:

```
% module use -a /u/your_username/privatemodules
```

Note: The -a option puts the new modulefile at the end of module avail output. Without it, the modulefile will be listed at the top.

Licensed Application Software

There are a few licensed applications available for your use on NAS systems. You can find these in the/nasa directory.

The available licenses were either purchased by NAS—with the justification that many users need the particular applications—or by users themselves. If you would like to use a licensed application that is not yet available on NAS systems, you may need to purchase the license yourself.

GNU Octave is an open-source program that you can use as an alternative to MATLAB. Its basic numerical functions are very similar to MATLAB, in terms of appearance and usage. Also, because the Octave language is similar to MATLAB, most MATLAB programs should be able to run on Octave. However, Octave toolboxes are different from MATLAB toolboxes. If you rely on any of the MATLAB toolboxes, you may not want to switch to Octave (or, you can make an appropriate modification).

For detailed information about Octave, see the GNU Octave home page.

For a comparison of the two programs, see Differences Between Octave and MATLAB on the GNU Octave wiki page.

Using Octave

The latest version of Octave, Version 4.0.0, includes a graphical user interface (GUI). All of the previous versions run only with a command-line interface.

To see a list of versions available on NAS systems, run:

```
%> module avail octave
```

To check the version you are using:

```
%> octave --version
```

To access a quick help guide:

```
%> octave --help
```

If you are using Octave Version 4.0.0 but you do not want to use the GUI, add the `--no-gui` option:

```
%> octave --no-gui
```

For comprehensive instructions on using Octave, see the GNU Octave documentation.

Demonstration

Octave's "look and feel" is similar to MATLAB. To see a demonstration, run:

```
>> demo waterfall
```

A waterfall plot is displayed.

Note: The Octave demonstration feature is not as powerful as that of MATLAB, but it shows you the similarities between the two programs.

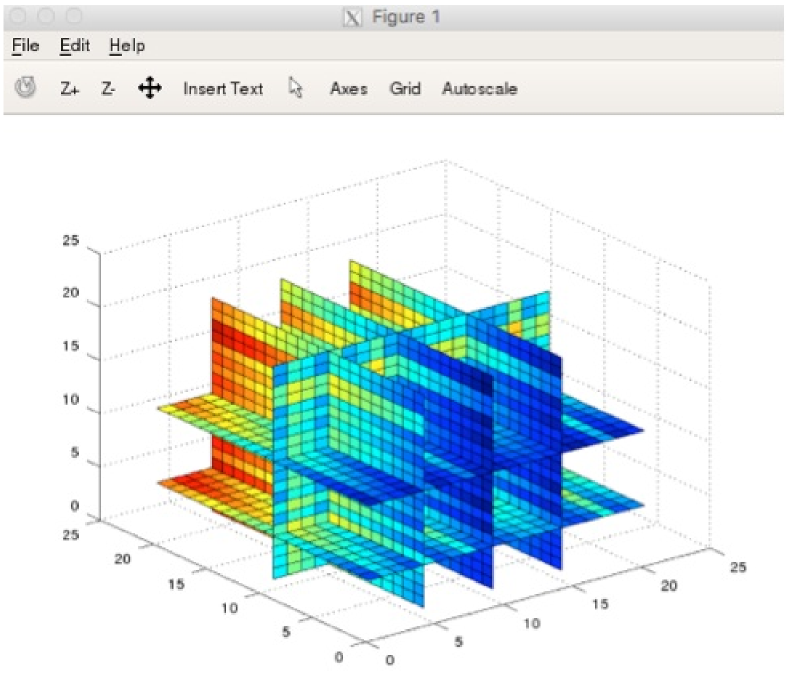
Example

Octave supports the same basic matrix operations and plotting as MATLAB. For example, if you run the following MATLAB code in Octave, the result will provide the same graphics as MATLAB.

MATLAB code:

```
x1 = -2*pi:pi/10:0;
x2 = 2*pi:pi/10:4*pi;
x3 = 0:pi/10:2*pi;
[x1,x2,x3] = ndgrid(x1,x2,x3);
z = x1 + exp(cos(2*x2.^2)) + sin(x3.^3);
slice(z,[5 10 15], 10, [5 12]);
```

Result in Octave:



Note: If there is a function in your MATLAB code that has not been implemented in Octave, a message will be displayed. For example:

```
warning: the 'delaunayTriangulation' function is not yet implemented in Octave

Please read `http://www.octave.org/missing.html' to learn how you can
contribute missing functionality.
warning: called from
    __unimplemented__ at line 524 column 5
error: 'delaunayTriangulation' undefined near line 1 column 6
```

If you want to run an application on Pleiades with a license that resides on a local license server (not directly accessible from outside your local network), you can use SSH tunneling, also known as port forwarding. The examples below show you how to establish port forwarding and run your application on either a Pleiades front end (PFE) or a Pleiades compute node.

Note: The examples assume that the hostname of your local license server islicense.abc.edu, which uses port 29810, and that you are connecting from your local system to pfe24. Be sure to replace these with the correct name and port number of your license server and the actual PFE you want to connect to. To find the hostname and port number of your local license server, check the setting of the LM_LICENSE_FILE environment variable on your local system or consult with your system administrator.

Step 1. Connecting to Pleiades

Use SSH to connect to pfe24 from your local system and establish remote port forwarding using the-R option, as follows:

- If you normally use one-step connection (with SSH passthrough):

  ```
  your_local_system% ssh -R 29810:license.abc.edu:29810 pfe24
  ```

- If you normally connect to Pleiades in two steps (connecting first to an SFE, and then to a PFE):

  ```
  your_local_system% ssh -R 29810:license.abc.edu:29810 sfeX
  (enter passphrase/passcode/password as needed here)
  sfeX% ssh -R 29810:localhost:29810 pfe24
  ```

where *sfeX* represents sfe[*6-8*].

In both cases, the port number listed *before* :license.abc.edu (the first 29810) is used by pfe24, while the port number listed *after* license.abc.edu: (the second 29810) is used by the license server.

Note: We highly recommend using the same port number for both the PFE and the license server, as shown in the example, and that you use the number set in the LM_LICENSE_FILE environment variable on your local system. However, if you choose to use a different port number for the PFE, be sure to avoid using numbers 0-1024 (root privilege required), 5900 (on Mac systems, reserved for some Apple remote desktop products), 59*xx* (reserved for VNC) and 6000-6063 (reserved for local X window server).

Step 2. Running Your Application

You can now run your application on the PFE or on a Pleiades compute node via an interactive PBS session.

Running Your Application on the PFE

In the same window where you connected to pfe24, set the environment variable LM_LICENSE_FILE to the port number you specified for pfe24 in your initial connection (i.e., the number *before* license.abc.edu), and then run your application:

```
pfe24% setenv LM_LICENSE_FILE 29810@localhost
pfe24% ./app
```

Note: You can run both commands in a separate window that has a new connection to pfe24. However, in order for the connection to the license server to remain active, the first connection window (with the remote port forwarding) must remain open.

Running Your Application on a Compute Node

In the same window where you connected to pfe24—or in a new window with a connection to any PFE—submit an interactive PBS job to get into a compute node. On the compute node (for example, r451i1n5), establish local port forwarding by using the SSH -L option. Both the compute node and pfe24 should use the same port you specified for pfe24 in your initial SSH connection (i.e., the number *before* license.abc.edu). Then, run your application.

```
pfe24% qsub -I -V -lselect=1:model=ivy
r451i1n5% ssh -fNL 29810:localhost:29810 pfe24
r451i1n5% setenv LM_LICENSE_FILE 29810@localhost
r451i1n5% ./app
```

Note: In the SSH command line, the -N option tells SSH to do port forwarding only, without executing a remote command. The -f option tells SSH to go to the background just before a command is executed.

For Applications that Require Two Ports

Consult with your system administrator for the correct port numbers to use, or check your local LM_LICENSE_FILE settings.

The following examples assume that the two ports needed are 29810 and 28500.

- To run the application on pfe24:

```
your_local_system% ssh -R 29810:license.abc.edu:29810 -R 28500:license.abc.edu:28500 pfe24
pfe24% setenv LM_LICENSE_FILE 29810@localhost:28500@localhost
pfe24% ./app
```

- To run the application on a compute node:

```
your_local_system% ssh -R 29810:license.abc.edu:29810 -R 28500:license.abc.edu:28500 pfe24
pfe24% qsub -I -V -lselect=1:model=ivy
r451i1n5% ssh -fNL 29810:localhost:29810 -L 28500:localhost:28500 pfe24
r451i1n5% setenv LM_LICENSE_FILE 29810@localhost:28500@localhost
r451i1n5% ./app
```

Gprof is a compiler-assisted performance profiler for C, Fortran, and Pascal applications running on Unix systems. You can use Gprof to help identify hotspots in your application where code optimization efforts may be most useful.

Gprof uses a hybrid of sampling and instrumentation, and provides the following information:

- The number of calls to each function and the amount of time spent there.
- Information about the caller-callee relationship.

Note: Gprof only measures the user code; it does not provide information on time spent in the kernel (such as system calls or I/O wait time).

The profiling data will be collected in a file called gmon.out, which will be generated at the end of a successful, uninterrupted run.

Gprof is available in the /usr/bin directory on Pleiades. To use this tool, follow the instructions in the sections below.

Compiling and Linking to Enable Profiling with Gprof

To enable profiling with Gprof, add one of the options shown below when you compile your code:

- With Intel compilers, add the -p option (alternatively you can add -pg, which is deprecated, but still works).
- With PGI compilers, add the -pg option.
- With GNU compilers, add the -pg option. You might also have to use the -O0 option, if you do not get meaningful profiling results when using higher levels of optimization.

Collecting Profiling Data

To collect profiling data, simply run your gprof-enabled executable the same way you would run a non-gprof-enabled executable. The data will be collected in a file called gmon.out.

OpenMP Applications

If you are using an OpenMP application, gprof does not generate per-thread profiling data. Only onegmon.out file is produced.

Note: If a file named gmon.out already exists in the directory, it will be overwritten.

MPI Applications

For MPI applications, if you use the Intel MPI library, no additional steps are required before you run your code. However, if you use the HPE MPT library, you also need to set the MPI_SHEPHERD variable as follows before you run the code; otherwise, the timing information will not be shown:

```
export MPI_SHEPHERD=true (bash)
setenv MPI_SHEPHERD true (csh)
```

Each MPI process will generate a profile with the same filename, gmon.out. These files will overwrite one another when they are written to a global filesystem. Therefore, to avoid this behavior and produce a profile with a distinct filename for each process, do:

```
export GMON_OUT_PREFIX=gmon.out (bash)
setenv GMON_OUT_PREFIX gmon.out (csh)
```

For example:

```
#PBS ...

module load comp-intel/2020.4.304
module load mpi-hpe/mpt.2.25
export MPI_SHEPHERD=true
export GMON_OUT_PREFIX=gmon.out

mpiexec a.out
```

This operation will generate a file called gmon.out.*pid*, where *pid* is the process ID of an MPI process. With *N* ranks, you should get *N* such files, with filenames differing only in their .*pid* extensions. You can then analyze an individual gmon.out.*pid* file or several at the same time.

Generating ASCII Gprof Output

You can use the gprof command to convert binary data in gmon.out into a human-readable format.

```
gprof [options] executable_name [gmon.out] [ > analysis.output ]
```

There are two types of output: *flat profile* and *call graph.*

The flat profile shows how much time your program spent in each function, and how many times that function was called. This profile helps to identify hotspots in your application. Hotspots are shown at the top of the flat profile.

The call graph shows, for each function, which functions called it; which other functions it called; and how many times the calls occurred. The call graph also provides an estimate of how much time was spent in the subroutines of each function, which can suggest places where you might try to eliminate function calls that use a lot of time.

Commonly Used gprof Command Options

Some common options are described here. Read **man gprof** for more information.

- -p prints a flat profile. For example:

  ```
  gprof -p a.out gmon.out.1001 gmon.out.1002
  ```

- -q prints a call graph. For example:

  ```
  gprof -q a.out gmon.out.*
  ```

- -s sums up the information from multiple profiling data files and produces a file called gmon.sum for analysis. For example:

  ```
  gprof -s a.out gmon.out.*
  gprof a.out gmon.sum > analysis.output
  ```

- -b omits verbose texts that explain the meaning of all of the fields in the tables.

IOT is a licensed toolkit developed by I/O Doctors, LLC, for I/O and MPI instrumentation and optimization of high-performance computing programs. It allows flexible and user-controllable analysis at various levels of detail: per job, per MPI rank, per file, and per function call. In addition to information such as time spent, number of calls, and number of bytes transferred, IOT also provides the time of the I/O and/or MPI call, and where in the file the I/O occurs.

IOT can be used to analyze Fortran, C, and C++ programs as well as script-based applications, such as R, MATLAB, and Python. The toolkit works with multiple MPI implementations, including HPE MPT and Intel MPI. It is available for use on Pleiades, Electra, and Endeavour. Basic instructions are provided below. If you are interested in more advanced analysis, contact User Services at support@nas.nasa.gov.

Setting Up IOT

To set up IOT, complete the following steps. You only need to do these steps once.

1. Add /nasa/IOT/latest/bin64 to your search path in your .cshrc file (for csh users) or .profile file (for bash users), as shown below. Be sure to add this line *above* the line in the file that checks for the existence of the prompt.

   For csh, use:
   ```
   set path = ( $path /nasa/IOT/latest/bin64 )
   ```

   For bash, use:
   ```
   PATH=$PATH:/nasa/IOT/latest/bin64
   ```

2. Run the iot -V command to check whether IOT is working. The output should be similar to the following example:

   ```
   % iot -V
   Using IOT install /nasa/IOT/v4.0.04/
         bin64/iot          v4.0.04 built Jun 14 2017 11:23:19
         lib64/libiot.so     v4.0.04 built Jun 14 2017 11:23:19
         libiotperm.so  v3.2.08 "Nasa_Advanced_SuperComputing" 5/11/2018 *
   ```

3. In your home directory, untar the /nasa/IOT/latest/ipsd/user_ipsd.tgz file:

   ```
   % tar xvzf /nasa/IOT/latest/ipsd/user_ipsd.tgz
   ```

   A directory called ipsd should be created under your $HOME directory.

4. Confirm that ipsd can be started:

   ```
   % ipsctl -A `hostname -s`
   No shares detected
   ```

5. Test IOT using the dd utility. First, create a directory called "dd" and change (cd) into it. Then, run the iot command as follows:

   ```
   % iot dd if=/dev/zero of=/dev/null count=20 bs=4096
   20+0 records in
   20+0 records out
   81920 bytes (82 kB) copied, 0.000123497 s, 663 MB/s
   ```

   In addition to the output shown above, you should also find a file called iot.*xxxxx*.ilz. The ILZ file is the output from the iot command.

Using IOT to Analyze Your Application

Once IOT is set up, follow these steps to analyze your application.

1. Create a configuration file that tells IOT what you want to instrument or monitor. You can use one of the following sample configuration files, which are available in the /nasa/IOT/latest/icf directory:

   trc_summary.icf
   : Use this file to start your first I/O analysis. This file provides a summary of information on the total counts, time spent, and bytes transferred for each I/O function of each file. For MPI applications, it also provides the same information obtained with mpi_summary.icf (described below).

   trc_interval.icf
   : In addition to the data collected by trc_summary.icf, this file provides more details for the read/write MPI function, per 1000-ms interval, including: the wall time when the calls occur; counts; time spent; and bytes transferred.

   trc_events.icf
   : In addition to the data collected by trc_summary.icf, this file provides the most details for each read/write function at the per-event level, including: the wall time when each call occurs; the time spent for the call; and the number of bytes transferred.

   mpi_summary.icf
   : Use this file to start your first MPI analysis. The file provides a summary of information such as the total count, time spent, and bytes transferred for all of the MPI functions called by the MPI ranks.

mpi_interval.icf
: In addition to the information collected by mpi_summary.icf, this file provides more details for the MPI functions, per 1000-millisecond (ms) interval, including: the wall time when the calls occur; counts; time spent; and bytes transferred.

mpi_events.icf
: This file provides the most details for each MPI function, at the per-event level, including: the wall time when each call occurs; the time spent for the call; and the number of bytes transferred.

2. Modify the `mpiexec` execution line in your PBS script to run IOT. For example, replace `mpiexec -np 100 a.out` with the following lines:

   ```
   set JOB_NUMBER=`echo $PBS_JOBID | awk -F. '{ print $1 }'`
    iot -m mpt -f cfg.icf -c '${HOST}':pfe22:`pwd`/a.out.collect.${JOB_NUMBER}.ilz \
    mpiexec -np 100 a.out
   ```

   This method will generate an ILZ file named a.out.collect.$.ilz.

   Another option is to simply use:

   ```
   iot -m mpt -f cfg.icf \
    mpiexec -np 100 a.out
   ```

   This method will generate an ILZ file named iot.*process_id*.ilz.
   TIP: When the -m option is enabled, the default value for -f is mpi_summary.icf, which will be located automatically in the /nasa/IOT/latest/icf directory.

For more information, see iot -h on the **IOT Options** and iot -M on the **IOT Layers** man pages.

Viewing the ILZ File

Once your ILZ file is generated, you can view the data with the Pulse graphical user interface (GUI) using one of the following methods. Pulse will read in the data from the file and organize it for easy analysis in the GUI.

Run Pulse on Your Local System (Recommended)

Follow these steps to run Pulse on your local system.

1. Download Pulse.jar and the ILZ file:

   ```
   your_local_system% scp pfe:/nasa/IOT/latest/pulse.d/Pulse.jar .
   your_local_system% scp pfe:/path_to_ilz_file/filename.ilz .
   ```

2. Run Pulse through Java:

   ```
   your_local_system% java -jar Pulse.jar filename.ilz
   ```

Note: Download the latest version of Pulse.jar from time to time, as enhancements may be added.

Run Pulse from a PFE

Log into a PFE, load a Java module, and run Pulse:

```
pfe21% module load jvm/jrel.8.0_121
pfe21% pulse filename.ilz
```

TIPS:

- Pulse will uncompress the ILZ file to 4-5 times its compressed size. If the uncompressed file gets very large, Pulse may run out of memory. If this happens, you can try to increase memory using the java -Xmx4g option, as follows:

  ```
  % java -Xmx4g -jar Pulse.jar filename.ilz
  ```

- While Pulse is reading the file, the filename in the GUI will appear in red text. You can stop it before Pulse consumes too much memory by right-clicking the filename and selecting **Stop Reading**.

Additional Documentation

IOT documentation provided by the vendor is available in the /nasa/IOT/Doc directory.

Two Tecplot products are available on NAS systems: Tecplot 360 and Tecplot Chorus.

Tecplot 360

Tecplot 360 is a computational fluid dynamics and numerical simulation visualization software program used for post-processing simulation results. Common tasks associated with post-processing analysis of flow solvers (for example, Fluent, STAR-CD, OpenFOAM) include:

- Calculating grid quantities (such as aspect ratios, skewness, orthogonality, and stretch factors)
- Normalizing data; deriving flow field functions (such as pressure coefficient or vorticity magnitude)
- Verifying solution convergence
- Estimating the order of accuracy of solutions
- Interactively exploring data through cut planes (slices through a region), iso-surfaces (3D maps of concentrations), particle paths (dropping an object in the "fluid" and watching where it goes)

The NAS Tecplot 360 license does not restrict the number of instances of Tecplot 360 that can be run concurrently.

TIP: If you are having problems running Tecplot through VNC, you can try running it with the -mesa option, which uses software rendering.

Tecplot Chorus

Tecplot Chorus is a simulation analytics framework that unites physics visualization with data management and analytics in a single environment.

Currently, one Tecplot Chorus license is available to use on Pleiades or NAS-supported desktop systems.

For more information, see:

- Tecplot product documentation
- Tecplot (Wikipedia)

Interactive Data Language (IDL) is a programming language for data analysis, visualization, and cross-platform application development. IDL combines tools for many types of projects, from "quick-look," interactive analysis and display to large-scale commercial programming projects.

NAS currently has IDL development licenses that allow 10 users to use the tool at the same time. Each license allows a single user to run multiple IDL sessions with the same session display simultaneously. Follow these steps to take advantage of this feature:

1. Use SSH to connect to a PFE.
2. Check the DISPLAY variable of the PFE, and launch multiple xterm sessions to run in the background. For example:

```
pfe25% echo $DISPLAY
pfe25:83.0

pfe25% xterm&
pfe25% xterm&
pfe25% xterm&
```

3. In each xterm window, check the DISPLAY variable to confirm that it has the same number that was shown in the original window. Load an IDL module, then launch the idl command. For example:

```
pfe25% echo $DISPLAY
pfe25:83.0
pfe25% module load idl/8.1
pfe25% idl
IDL>
```

In the example above, a single user occupies only one IDL license rather than three licenses by ensuring that the IDL sessions are running in three separate xterm windows that all have the same DISPLAY number.

Use one of the following methods to view how many licenses are currently in use:

- Run /u/scicon/tools/bin/check_licenses -i
- Load an IDL module and run the lmstat -a command

WARNING: You must quit IDL and release the license when you are done. Keeping the license in use when you are not actively using IDL prevents others who need access to IDL from doing useful work.

If you cannot use IDL because all licenses are currently being used, try again at a later time. If you notice that multiple licenses are occupied by the same user, please contact the NAS Control Room at (800) 331-8737, (650) 604-4444, or by email at support@nas.nasa.gov.

For more information, see:

- NAS IDL License Policy
- IDL Documentation
- IDL Programming Language (Wikipedia)

In order to effectively utilize the limited number of Interactive Data Language (IDL) licenses on NAS-supported workstations or supercomputing systems:

- Each user is limited to a single interactive IDL license.
- When using the IDL command line, after four hours of being idle and not running a job, the license will be revoked.
- If users are waiting for licenses, a IDL process that has been idle for more than ten minutes may be terminated to revoke its license and make it available to another user.

MATLAB is a numerical computing environment and programming language, created by MathWorks, that enables you to easily manipulate matrices, plot functions and data, implement algorithms, create user interfaces, and interface with programs in other languages. Although MATLAB specializes in numerical computing, an optional toolbox interfaces with the Maple symbolic engine, allowing it to be part of a full computer algebra system.

A total of 16 licenses are available to use on either NAS HPC systems or NAS-supported desktop systems. To find out how many licenses are currently in use, run:

```
/u/scicon/tools/bin/check_licenses -m (view general licenses only)

/u/scicon/tools/bin/check_licenses -M (view general licenses and compiler/toolboxes)
```

Then, load a MATLAB module:

```
% module load matlab/2016b
```

See the following websites for more information about MATLAB software:

- MATLAB documentation (MathWorks)
- MATLAB (Wikipedia)

See the following Knowledge Base articles for more information about NAS MATLAB licenses:

- MATLAB License Policy
- Using NAS MATLAB Licenses

NAS currently has 16 MATLAB licenses available. To avoid overwhelming the license server, and to reduce contention for the limited number of licenses, PBS batch jobs running on Pleiades are not allowed to query the MATLAB license server. Therefore, in order to run a MATLAB script in a batch job on Pleiades, you must first compile the script. Compiled executables do not access the license server.

Note: It is not necessary to compile your MATLAB scripts for batch jobs running on the LDANs or Endeavour—on these systems, there is less risk of overwhelming the MATLAB license server because fewer hosts query the server. Also, you can still run interactive MATLAB jobs on Pleiades without compiling the scripts (it is easier to debug the scripts if they are not compiled into executables).

Steps for Compiling a MATLAB Script

This method is the simplest way to compile a MATLAB script into a standalone executable. The steps are described using the following sample script, findR.m:

```
function R = findR
% findR - given data T, calculate R
%
T = 10;
R = .5*(-9.8)*T
% end of MATLAB code
```

Complete these steps to compile the sample script:

1. Load the modules:

   ```
   % module load matlab
   % module load gcc/4.7.3 # or later
   ```

2. If this is the first time you are using MATLAB to create an executable, run:

   ```
   % matlab -nodisplay
   >>mbuild -setup
   >>exit
   ```

3. Compile the script into an executable. At the Linux or MATLAB prompt, run:

   ```
   % mcc -m findR.m -o findR
   ```

   Two files are created:
   - An executable: findR
   - A shell script: run_findR.sh
4. To run the executable, add the following lines to your PBS script or run them at the Linux prompt:

   ```
   % module load matlab/2016b
   % module load gcc/4.7.3
   ./run_findR.sh $MATLAB
   ```

   The output will be:

   ```
   % ./run_findR.sh $MATLAB
   ------------------------------------------
   Setting up environment variables
   ---
   LD_LIBRARY_PATH is
   .:/nasa/mw/2016b/runtime/glnxa64:/nasa/mw/2016b/bin/glnxa64:/nasa/mw/20
   14b/sys/os/glnxa64:/nasa/mw/2016b/sys/opengl/lib/glnxa64
   R =
     -49
   ```

Note: If you have several MATLAB scripts, where master.m calls y.m, and y.m calls z.m, make sure that master.m is the first file listed in the command line when you compile the script:

```
% mcc -m master.m y.m z.m
```

In this case, the resulting files are:

- Executable: master
- Shell script: run_master.sh

In order to effectively utilize the limited number of MATLAB licenses on NAS-supported workstations or supercomputing systems:

- Each user is limited to a single interactive MATLAB license.
- When using the MATLAB GUI, after four hours of being idle and not running a job, the license will be revoked.
- Start a second MATLAB GUI on the same host to compile or use toolboxes. Exit the second GUI when you are done with the license to immediately free up the compiler/toolbox license.
- If users are waiting for licenses, a MATLAB process that has been idle for more than ten minutes may be terminated to revoke its license and make it available to another user.

For more information, including alternatives to using MATLAB licenses, see Using NAS MATLAB Licenses.

A limited number of MATLAB licenses are available on NAS-supported workstations and supercomputing systems. To check current MATLAB license usage, run the check_licenses command as follows:

```
/u/scicon/tools/bin/check_licenses -m (view general licenses only)

/u/scicon/tools/bin/check_licenses -M (view general licenses and compiler/toolboxes)
```

Important Usage Information

Please note the following information when you use a NAS MATLAB license:

- When the MATLAB compiler/toolbox is run from the GUI, the license is held by the user until the GUI is closed.
- When the MATLAB command-line tools (such as the compiler) are used, the license is held by the user for 30 minutes after the last use. Using the tool again restarts the timer.
- If multiple MATLAB licenses are checked out by a single user, the newest MATLAB process will be terminated.
- The single-license limit only applies to the primary MATLAB license. Licenses such as the compiler or toolboxes do not count toward the limit.

See the MATLAB License Policy for more information.

Alternatives to Using NAS MATLAB Licenses

Alternatives to using MATLAB licenses include:

- Compile non-interactive MATLAB jobs so they do not require a license. See Compiling MATLAB Scripts into Executables to Reduce the Use of Licenses.
- If you have access to a local MATLAB license that you are entitled to use remotely, you can utilize it byrunning applications on Pleiades with your local license.
- Use GNU Octave, an open source alternative to MATLAB. GNU Octave is available on Pleiades. To access it, run the following commands:

  ```
  module avail octave

  ----------------------- /nasa/modulefiles/spack/gcc-4.8 ------------------------
  octave/4.2.1

  module load octave/4.2.1
  octave
  ```

UPDATE IN PROGRESS: Starting with version 2.17, **SGI MPT** is officially known as **HPE MPT**. Use the command module load mpi-hpe/mpt to get the recommended version of MPT library on NAS systems. This article is being updated to reflect this change.

MATLAB parametric studies and neural network applications can typically be run in parallel for higher productivity. However, because NAS does not have a license for MATLAB's Parallel Computing Toolbox and only 16 MATLAB licenses are available, NAS users often run these applications serially.

There are two ways to run multiple MATLAB instances in parallel without using multiple licenses:

- Invoking multiple MATLAB instances in PBS interactive mode
- Running MATLAB in parallel with MPI and Fortran

Each license is good for one node and one user, so you can run MATLAB on multiple cores on the same node using one license. To check the availability of MATLAB licenses, run the check_licenses tool as follows:

```
pfe% /u/scicon/tools/bin/check_licenses -m
```

Invoking Multiple MATLAB Instances in PBS Interactive Mode

Use this method if you want to execute a MATLAB function multiple times, and the function's input parameter can be created with simple arithmetic. In this case, you can easily invoke MATLAB in a for loop, as described below. (If the input parameter does not follow a simple arithmetic pattern, see Running MATLAB in Parallel with MPI and Fortran.)

To reduce contention for the limited number of licenses, PBS batch jobs are not allowed to query the MATLAB license server. However, for a simple parametric study like the following example, you can use PBS interactive mode.

Example

In this example, a MATLAB function called sampleFn.m is calculated 24 times with input parameters 2, 4, 6, ..., 48. The sampleFn.m function runs all 24 instances in parallel on the same Haswell node.

First, submit a request for one Haswell node:

```
%> qsub -I -l select=1:mpiprocs=24:model=has -q devel -l walltime=2:00:00
qsub: waiting for job 8022657.pbspl1.nas.nasa.gov to start
Job 8022657.pbspl1.nas.nasa.gov started on Mon Feb 03 15:55:10 PST 2020
.....
PBS r509i0n17 51>
```

From the compute node (r509i0n17), run the following commands to start 24 instances in parallel:

```
PBS r509i0n17 51> source /usr/local/lib/init/global.cshrc
PBS r509i0n17 51> module purge
PBS r509i0n17 51> module load matlab/2017b
PBS r509i0n17 51> foreach i (`seq 1 24`)
foreach?  @ j = $i * 2
foreach? matlab -nodesktop -nodisplay -nosplash -r "sampleFn(${?j?});exit" > out__$i < /dev/null &
foreach? end
PBS r509i0n17 51> wait
```

All processes should return at the same time. There will be 24 files with the name out__*.

Note: Because there are 24 cores on a Haswell node, no more than 24 MATLAB instances should be run at the same time.

Sample MATLAB function: sampleFn.m

The sampleFn function called in the example above could be any MATLAB function. For example, it might look like this:

```
function D = sampleFn(T)
%
% sampleFn - given data T, return D = 3x3 matrix with entries T
% D = is 3x3 matrix with entries T
%
n=3;
D = T*ones(n,n)
```

Running MATLAB in Parallel with MPI and Fortran

Use this method if you want to execute a MATLAB function multiple times and the input parameter does *not* follow a simple arithmetic pattern. (Otherwise, use the PBS script method described above.)

This method uses an MPI/Fortran program to run multiple MATLAB instances in parallel. Each MPI process in the Fortran program activates the MATLAB engine and calls the MATLAB function you intend to run. Due to differences in the ways Fortran and MATLAB pass arguments, the Fortran code also needs to call additional MATLAB functions to properly transfer argument values.

The MPI/Fortran program is then compiled via MATLAB's mex script and linked with MATLAB and MPI libraries. You can run the resulting executable as a typical MPI executable using the mpiexec command through a PBS job.

Sample MPI/Fortran Code

Suppose you want to call the sampleFn(s) routine described in the script method above, but you want to pass in a real number whose value "s" cannot be obtained by simple arithmetic. Instead, the value will be obtained by a Fortran (or C) routine.

Note: The value of "s" can be any data type.

The sample MPI/Fortran code test.F demonstrates how this method works. The key actions in test.F are to activate the MATLAB engine, facilitate transfer of input, call sampleFn(), and transfer output, as shown in the following steps. Each step indicates a specific line in the test.F code.

1. Line 41 - activate the MATLAB engine:
   ep = engOpen('matlab ')
2. Line 49 - define the Fortran pointer of a MATLAB scalar:
   sPtr = mxCreateDoubleMatrix(1, 1, 0) ! 1x1 matrix is a scalar
   This scalar is the "s" in the MATLAB function.
3. Line 52 - obtain the parameter to feed into the MATLAB code, sampleFn(s):
   k = IRAND(rank)
   In this example, a random integer value "k" for each MPI rank is obtained from a random number generator. (This Fortran value "k" will be converted to value "s" in the MATLAB code.)
4. Line 54 - assign the Fortran value "k" to the pointer sPtr:
   call mxCopyReal8ToPtr(dble(k), mxGetPr(sPtr), 1)
   Note that each MPI process has its own value for "k".
5. Line 59 - call sampleFn():
   if (engEvalString(ep, 'D = sampleFn(s);') .ne. 0) then
   Each MPI process executes the MATLAB function that has output D. Note that the value of D can be any data type and dimension. In this case, D is a real nxn matrix.
6. Line 65 - assign the MATLAB value D to Fortran pointer DPtr:
   DPtr = engGetVariable(ep, "D")
7. Line 66 - instruct each MPI process to obtain the value of the nxn matrix D:
   call mxCopyPtrToReal8(mxGetPr(DPtr), D, n*n)

Fortran Code: test.F

Scroll inside the code box to see the whole code. The command lines noted in the steps above are highlighted.

```
 1 #include "fintrf.h"
 2 C
 3 #if 0
 4 !
 5 !     test.F
 6 !     .F file need to be preprocessed to generate .for equivalent
 7 !
 8 #endif
 9 !
10 !     test.F
11 !
12 !     This is a simple program modified from a sample code that
13 !     illustrates how to call the MATLAB Engine functions.
14 !
15 ! NASA Ames Research Center
16 !======================================================================
17 ! 2015/7/7
18
19       program main
20       implicit none
21       include 'mpif.h'
22 !-----------------------------------------------------------------------
23 !     (pointer) Replace integer by integer*8 on 64-bit platforms
24 !
25       mwpointer engOpen, engGetVariable, mxCreateDoubleMatrix
26       mwpointer mxGetPr
27       mwpointer ep, DPtr, sPtr
28 !-----------------------------------------------------------------------
29 !
30 !     Other variable declarations here
31       integer, parameter :: n=3, M=12
32       double precision D(n,n), Darray(n,n,M), s
33       integer engPutVariable, engEvalString, engClose
34       integer i,k, temp, status, size , rank, ierror
35 !
36       ! MPI initilization
```

```
      call MPI_INIT(ierror)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)

     ep = engOpen('matlab ')
!
      if (ep .eq. 0) then
         write(6,*) 'Can''t start MATLAB engine'
         stop
     endif
C
!    Define the pointer of a Matlab scalar
    sPtr = mxCreateDoubleMatrix(1, 1, 0)    ! 1x1 matrix is scalar

!    Obtain the value of k from a routine (here it is just random generator
    k = IRAND(rank)
!      Copy Fortran value k to Matlab array mxGetPr(sPtr)
     call mxCopyReal8ToPtr(dble(k), mxGetPr(sPtr), 1)
       status = engPutVariable(ep, 's', sPtr)
       if (status .ne. 0) write(*,*)"FAIL!!! engPutVariable s"
!
!      Compute via matlab function : sampleFn
     if (engEvalString(ep, 'D = sampleFn(s);') .ne. 0) then
        write(6,*) 'engEvalString failed'
        stop
      endif
!
!     Bring pointer back to Fortran
     DPtr = engGetVariable(ep, "D")
     call mxCopyPtrToReal8(mxGetPr(DPtr), D, n*n)

!     The MPI master can gather all D's to Darray
!     via MPI_Send/Recv or MPI_Gather or MPI_Reduce
!     Darray(:,:,k) <= D(:,:)
!
     print *, 'Print results created by Matlab :'
       do i=1,n
       write(*,*) D(:,i)
       enddo
       write(*,*) " "
!
!
     call mxDestroyArray(sPtr)
     call mxDestroyArray(DPtr)
     status = engClose(ep)
!
     if (status .ne. 0) then
       write(6,*) 'engClose failed'
       stop
      endif
      call MPI_Finalize(ierror)
!
      stop
      end program main
```

Compiling the MPI/Fortran Code

This example uses mex, /usr/bin/gfortran, matlab/2016b, and mpi-sgi/mpt.2.12r26. to compile the program. You can choose other versions of compiler, MATLAB, and MPT.

To compile, run:

```
mex -v LDFLAGS="-lmpi" FFLAGS=" -fexceptions -fbackslash \
    -I/nasa/mw/2016b/extern/include -I/nasa/sgi/mpt/2.12r26/include" \
    -f /nasa/mw/2016b/bin/engopts.sh test.F
```

The resulting executable is called test.

If you want to use the Intel Fortran compiler, add FC=ifort and replace -fbackslash with -assume bscc in the mex command, as follows:

```
mex FC=ifort -v LDFLAGS="-lmpi" FFLAGS=" -fexceptions -assume bscc \
    -I/nasa/mw/2016b/extern/include -I/nasa/sgi/mpt/2.12r26/include" \
    -f /nasa/mw/2016b/bin/engopts.sh test.F
```

Running the Executable

Within a PBS script or an interactive PBS session, load the necessary MATLAB, MPI and compiler modules, and set the LD_LIBRARY_PATH environment variable to the correct setting before executing mpiexec as follows:

```
% setenv LD_LIBRARY_PATH  "/nasa/mw/2016b/bin/glnxa64:   \
    /nasa/mw/2016b/sys/os/glnxa64:$LD_LIBRARY_PATH"

% mpiexec -np 4 ./test
```

In this example, four MATLAB instances are running.

IMPORTANT: As a courtesy to other MATLAB users, please do not use more than two MATLAB licenses (i.e., using more than 2 nodes) per PBS job.